Joel Coelho Pinheiro

# iTrading

**Joel Coelho Pinheiro**

# iTrading

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Doutor João Paulo Silva Barraca, Professor convidado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

**o júri / the jury**

| | |
|---|---|
| presidente / president | **Prof. Dr. Joaquim Arnaldo Carvalho Martins**<br>Professor Catedrático da Universidade de Aveiro |
| vogais / examiners committee | **Prof. Dr. Daniel Ferreira Polónia**<br>Professor Auxiliar Convidado da Universidade de Aveiro |
| | **Prof. Dr. Diogo Nuno Pereira Gomes**<br>Professor Auxiliar da Universidade de Aveiro (orientador) |

**agradecimentos / acknowledgements**

Quero agradecer a todos os que me acompanharam nestes últimos meses, incluindo os meus pais, a minha irmã, a minha namorada, e todos os meus colegas e amigos, de Aveiro e de Águeda. Queria também agradecer ao Professor Doutor Diogo Gomes e ao Professor Doutor João Paulo Barraca por me guiarem e orientarem neste percurso sempre que precisei.

Não teria conseguido sem todos vocês. Obrigado.

**Palavras-Chave**

negociação, intermediação financeira, lua, negociação automática, backtesting, arquitetura, software, multi-plataforma, negociação, algorítmico, embutido, visualização, gráficos, api, ib, gestão, dados, mercado.

**Resumo**

A Internet permitiu revolucionar várias áreas económicas graças à facilidade com que é possível distribuir informação e comunicar entre entidades. Existem ainda áreas onde a Internet não só revolucionou os mercados financeiros, como levou à criação de novos mercados, permitindo o acesso desses mercados a novas entidades. Neste contexto, o aparecimento de mercados de negociação de bens e serviços em tempo-real é paradigmático. As bolsas de valores, mercados primários, correctores de apostas, entre outros, viram o seu modelo de funcionamento alterado pela Internet. Estes mercados passaram a negociar em permanência, pelo que, o número de ordens financeiras subiu tão exponencialmente que é actualmente necessário recorrer a complexas plataformas de transações. Hoje em dia existem inúmeras aplicações de negociação em tempo-real para os diversos mercados, umas nativas (domínio de plataformas Microsoft) e outras Web (limitações ao nível de tempo de resposta e das capacidades gráficas). Um aspecto comum a todas elas é o facto de se centrarem na negociação electrónica de ordens emitidas de forma explícita por humanos e ter apenas automatismos para situações de controlo de prejuízo (via triggers).

Esta dissertação pretende, por isso, estudar o desenvolvimento de uma nova geração de aplicações de trading que incluam um ambiente de programação embutido na própria aplicação, automação de negociação e backtesting. De forma a colmatar a inexistência deste tipo de aplicações em ambientes não Windows, pretende-se que a mesma seja desenvolvida para ambientes Linux, OSX e Windows.

**keywords** trading, financial, brokerage, lua, automation, backtesting, software architecture, cross-platform, embedded, algorithmic, graphics visualization, ib, api, market, data, ticks.

**Abstract**

The Internet brought a revolution to several economic areas because it facilitated the distribution of information and communication between entities. In this context, the emergence of online trading markets of goods and services is paradigmatic. Markets started to negotiate continuously and the number of financial orders rose exponentially as it is currently necessary to employ complex transactions platforms. Today, there are numerous applications of online trading, some are native (limited to platforms such as Microsoft OS), others are Web-based (latency issues).

This dissertation presents the development of a new generation of trading applications that includes an embedded programming environment in the application itself, trading automation and backtesting. It was developed as a multi platform application for Linux, OSX and Windows platforms.

# Contents

# List of Figures

# List of Tables

# Listings

# Glossary

**AJAX** Asynchronous Javascript and XML.

**API** Application Programming Interface.

**AT** Algorithmic Trading.

**CEF** Chromium Embedded Framework.

**CFD** Contract For Difference.

**CSS** Cascading Style Sheets.

**DOM** Document Object Model.

**DSL** Domain-Specific Language.

**GUI** Graphical User Interface.

**HTML** HyperText Markup Language.

**IB** Interactive Brokers.

**IRC** Internet Relay Chat.

**JS** JavaScript.

**JVM** Java Virtual Machine.

**LLVM** Low Level Virtual Machine.

**MATLAB** Matrix Laboratory.

**NASDAQ** National Association of Securities Dealers Automated Quotations.

**OS** Operating System.

**OSX** Mac-OS.

**OT** Online Trading.

**PC** Personal Computer.

**SVG** Scalable Vector Graphics.

**TWS** Trader Workstation.

**UI** User Interface.

**UK** United Kingdom.

**VM** Virtual Machine.

# 1 Introduction

## 1.1 Contextualization

The financial market is influenced by the changes and fluctuations of the economy. Additionally, it has grown as one of the principal markets of any country in the globe. Nowadays, the financial market performance is regularly considered as a barometer of economy. Furthermore, if one part of the globe gets stock market changes, other parts will change too. Based on this point, financial market growth can also predict the future of financial market performance worldwide [1].

Financial market growth can be directly associated with the growth of money markets. Money has risen from the days of the barter economy to facilitate the immediate exchange of goods and services. It has developed a usual way of exchange in the contemporary world. Banking has came as one of the primary institutions in today's world. Banking services have become more diversified with mortgage banking and investment banking coexisting with the general mode of banking services. Moreover, Financial market growth can be outlined to the growth of the insurance and derivates markets which have attained astounding importance in the recent years [2].

Financial markets pull funds from investors and pipe them to corporations. They consequently allow corporations to finance their operations and obtain growth. Money markets allow firms to borrow funds on a short-term funding to back expansion. Without financial markets, borrowers would have trouble getting lenders themselves. Entities like banks can help in this process. Banks receive deposits from those who have the capital to save. They can then lend money from this funds of deposited money to those who seek to borrow. More detailed actions than a simple bank deposit require markets where lenders and their agents can meet borrowers and their representatives and where existing borrowing or lending commitments can be sold on to other parties. A good example of a financial market is a stock exchange [3].

Above all, technology had invariably repercussions on financial markets. The Internet is a multi-purpose, multi-point digital interactive global telecommunications network. In its essence, the internet facilitates multi-point data flows and all the methods that are based on data flows. Financial intermediation and financial markets are based on the exchange of data. In fact, a transaction of exchange of any financial instruments, including cash, equities, bonds

and their derivates, is just a reporting of altered digital information. Last, but not the least, since the Internet is actually a global network, everything can be done transversely through national boundaries, as well as country and local authorities [4].

Improvements to existing processes and markets are some of the consequences made by Internet. Often the amount of improvement can be so significant that it has consequences for market structure. A typical example of an enhancement of an existing process is the exclusion of the middleman, broker, in sending orders to financial markets. Before the Internet, this was only achievable by using the telephone. Internet allows it to be done much more efficiently through a close connection to an electronic system. It brought an ample availability of information both about current rates and past performance as well as various instruments to analyse it.

An example of a radical enhancement is that Internet has created a huge pressure on removing price discrimination based on geography or political borders, as well as in growing price competition among providers of regulated goods. The Internet's capacity to allow the collection of pricing information from dozens of sellers decreases search costs immensely, increases price competition among different providers. The Internet also introduce new processes, goods and cooperation that were not imaginable before. In particular, digital goods can be distributed over the Internet [4].

This has been designated the "democratization of trading process" [5]. Tools that were available only in trading rooms are now widely available. For the astute and wise trader, these are very important and improved the playing field. For the foolish or the uninformed, the Internet makes it easier to lose money. Overall, it is worth perceiving that the wide availability of speedy action trading technology has grown market volatility [4].

Algorithmic Trading (AT) is a dramatic case of this far-reaching technological revolution. Many market members now employ AT, generally defined as the use of computer algorithms to automatically make specific trading decisions, submit orders and execute those orders after submission. From a starting point near zero in the mid-1990s, AT is thought to be held by as much as 73% of the trading amount in the United States in 2009 [6].

Technical indicators are mathematical estimations based on a trading instrument's past and current value and/or volume activity. Technical analysts use this data to appraise past performance and to prognosticate future rates. Indicators do not exactly provide any buy and sell signals; a trader needs to interpret the signals to determine trade entry and exit points that adapt to his or her own different trading style. Various different types of indicators exist, including those that interpret the trend, momentum, volatility and volume [7].

Because of the exponential growing of financial markets, today there are a lot of Online Trading (OT) applications to diverse markets. Several investors and sharp traders use professional trading indicators to identify high chance trade entry and exit points. A lot of indicators are accessible on most trading platforms, therefore, it is easy to use too many indicators or to use them inefficiently. It is critical how to select multiple indicators, how to bypass information overload and how to optimize indicators to most efficiently take benefit of these technical analysis tools [7].

Multicollinearity is a mathematical name that refers to the counting of the very information. This is a general problem in technical analysis that occurs when the identical types of indicators are applied to one chart. The results produce repetitive signals that can be misleading. Some traders deliberately apply multiple indicators of the equal type, in the expectations of finding evidence for an expected price move. In truth, however, multicollinearity can make other variables seem less important and can make it tricky to truly judge market circumstances [7].

To bypass the difficulties associated with multicollinearity, traders must select indicators that work fit with, or supplement, each other without providing repetitive results. This can be accomplished by applying different types of indicators to a chart. A trader could use one momentum and one trend indicator; for instance, a Stochastic oscillator (a momentum indicator) and an Average Directional Index (a trend indicator). Note how the indicators provide different information. Since each provides a different interpretation of market conditions, one may be used to confirm the other [7].

Since a trader's charting platform is his or her door to the markets, it is essential that the charts enhance and not hinder, a trader's market analysis. Simple to read charts and workspaces can increase a trader's situational awareness, enabling the trader to quickly interpret and answer to market activity. Most trading software allows for a great level of customization regarding chart color and design. Setting up cleanly and visually appealing charts and workspaces help traders use indicators effectively [7].

It's essential to see that technical analysis is based on probabilities rather than facts. There is no sequence of indicators that will accurately predict the markets' moves 100% of the time. While also several indicators, or the incorrect use of indicators, can shade a trader's view of the markets, traders who use technical indicators correctly and effectively can more accurately pinpoint high-probability trading setups, improving the odds for a successful investment [7].

## 1.2 Motivation

Nowadays, there are several OT applications for different markets. Some are native applications, Microsoft-oriented platforms mostly, which to run in a different Operating System (OS), need to be emulated. Additionally, their graphics are inadequate and deprecated. Other OT applications are web applications with obviously latency issues. A common aspect of these solutions is based on the fact that they centralize their means on electronic emitted orders, in an explicit way, by humans. They have only automation methods for loss control via triggers. Other solutions have AT functionality, but they lack a backtesting environment where traders can analyse their scripts success based on the past historical market data. Since the majority of people negotiating in financial market come from a financial/economy background and probably lack strong programming skills, the language used for AT needs to be simple to understand and to use.

This dissertation, intends to to develop a new generation of trading applications (iTrading) that include an embedded programming environment (Algorithmic Trading). Bearing in mind the inexistence of these types of application in non-Windows environments it is expected that

this software should be developed for Linux and OSX, besides Windows environments.

A fundamental distinction between a traditional investment management process and a quantitative process is the opportunity of backtesting a quantitative investment strategy to understand how it would perform in the past. It was intended to develop a backtesting strategy to evaluate the success of the scripts developed. Low-latency activity is the strategy that reacts to market events in the millisecond environment. As a trading platform, iTrading orders and ticks should be sent and received with the minimum latency possible.

To contend the lack of dynamic and flexible financial graphics, the resulting software should have good financial features regarding data analysis (e.g., indicators, interactive graphics). In order to make this possible, several technologies should be examined to get the most modern tools of financial trading graphics.

In terms of Securities law, the financial contract is an arrangement that takes the form of an individually negotiated contract, agreement, or option to buy, sell, lend, swap, or repurchase, or other similar individually negotiated transaction commonly entered into by participants in the financial markets. As a proof of a concept, it is proposed to include in iTrading the most popular financial contracts: Stocks, Forex, Contract For Difference (CFD)s, Combo, Futures and Options.

One of the most enduring sayings on Wall Street is "Cut your losses short and let your winners run" [8]. Obtaining a capital loss before it gets out of hand divides successful investors from the others. In this prototype, there will be the possibility to define a contract in the following orders: Market, Limit, Stop and Stop Limit.

## 1.3 Structure of the Dissertation

This thesis is divided into six chapters, with the first and last containing the introduction and final conclusion, respectively.

Chapter 2 presents the State of the Art. Three main topics are addressed namely Algorithmic Trading, Low-latency Trading and Backtesting with four tools examined: Excel; MATLAB; TradeStation and MT4. Afterwards, it were explored three algorithmic trading languages: Lua, Python and MQL4. In the next section it was made a comparison between the most popular trading solutions giving relevance to MetaTrader and ProTrader. Finally it were explored two financial brokers, Interactive Brokers and MB Trading Broker.

Chapter 3 introduces the proposed solution. Some of the available technologies were explored in order to get an adequate one to the proposed solution. After, it was described the functional requirements the system must cover and the visual representation of the application (Wireframes). Next, it is detailed the architecture of the solution. Finally, there is na mindmap representation of the entire SotA and Proposed Solution.

Chapter 4 details the implementation that was made to develop the prototype, installation

guide and the iTrading download web page.

Chapter 5 describes the evaluation process (questionnaire), visual results and profiling.

Finally, in chapter 6 is described the Contributions that were made during this dissertation and the Future Work which describes the next development phase.

At the end of each chapter, a summary was made, resuming and emphasizing the most relevant topics and ideas. After the thesis's conclusions, an annex section will be featured, composed of topics that provide further information on certain overviewed subjects.

# 2 State of the Art

This chapter describes the State of the Art. Three main topics are addressed namely Algorithmic Trading, Low-latency Trading and Backtesting with four tools examined: Excel; MATLAB; TradeStation and MT4. Afterwards, it were explored three algorithmic trading languages: Lua, Python and MQL4. In the next section it were compared the most popular trading solutions giving more importance to MetaTrader and ProTrader. The remaining solutions weren't so fully explored as the others due to the lack of literature. Finally it were explored two financial brokers, Interactive Brokers and MB Trading.

## 2.1 Algorithmic Trading Impact

Algorithmic trading adds an adequate price discovery process via the exclusion of triangular arbitrage events (buying in one market and simultaneously selling in another, profiting from a temporary difference) and the quicker incorporation of macroeconomic news into the price [9].

The adoption of algorithmic trading, where computers monitor markets and control the exchanging process at high frequency, has become popular in larger financial markets in recent years, rising in the U.S. equity market at the end of the 1990s. After the introduction of algorithmic trading, there has been public interest in understanding the potential repercussions it may have on market dynamics. While some have highlighted the potential for higher efficient price discovery, others have expressed interest that it may drive to higher adverse selection costs and unnecessary volatility. Despite the general interest, formal empirical research on algorithmic trading has been short, essentially because of a lack of data in which algorithmic trades have been classified. There is a study [9] that indicates the effect algorithmic ("computer") trades and non-algorithmic ("human") trades have on the informational effectiveness of foreign exchange prices. It is the first formal empirical study on the subject in the foreign exchange market. They rely on a novel dataset consisting of many years (October 2003 to December 2007) of "minute-by-minute trading data from Electronic Broking Services (EBS) in three currency pairs: the euro-dollar, dollar-yen and euro-yen". The data express a majority of interdealer transactions over the globe. A significant feature of the data is that, on a minute-by-minute frequency, the volume and direction of individual and computer trades are explicitly distinguished, enabling to measure their respective impacts at high frequency.

The research highlights two essential features: algorithmic trading's improvement in speed

over human trading and the possibly high correlation in algorithmic trader's strategies and actions. There is no consensus on the effect these two features may have on the informativeness of prices. There is an investigation [10] that claims that algorithmic traders' speed advantage over humans. Their capacity to react more promptly to public information than humans has a positive effect on the informativeness of prices. In their theoretical principles, algorithmic traders are better informed than humans and use market orders to employ their information. Given these hypotheses, the authors show that the presence of algorithmic traders makes asset prices more informationally valuable, but their trades are a source of adverse selection for those who give liquidity. These authors argue that algorithmic traders contribute to price discovery because once price inefficiencies exist, they quickly make them disappear. An individual can also claim that better informed algorithmic traders who specialize in providing liquidity make prices more informationally efficient by posting quotes that reflect new information quickly and therefore limit arbitrage opportunities from occurring in the first place.

In opposition to these positive aspects of algorithmic trading, Foucault, Hombert and Rosu [11] tell that in a world with no asymmetric data, algorithmic traders' speed advantage does not increase the informativeness of prices and just increases adverse selection costs. Jarrow and Protter [12] claim that both characteristics of algorithmic trading, the speed gain above human traders and the potential commonality of trading strategies with algorithmic traders, will have an adverse effect on the informativeness of prices. In their ideal model, algorithmic traders, triggered by a common signal, make the same trade at the same time. Algorithmic traders collectively perform as one big trader, create price momentum and consequently cause prices to be less informationally efficient. Algorithmic traders starting the same trade at the same time causes a crowding effect, which in turn pushes prices further away from fundamentals [9]. Stein [13] also highlights this crowding effect in the circumstances of hedge funds simultaneously implementing "convergence trade" strategies. In contrast, Oehmke and Kondor [9] claim that the higher the number of traders who implement "convergence trade" strategies the more efficient prices will be.

## 2.2 Low-latency Trading

While there are several definitions for the term "latency" it is viewed as the time it takes to get about an event (e.g., a change in the bid), produce an answer and have the exchange act on the reply. Low-latency activity is the strategy that reacts to market events in the millisecond environment, the symbol of exclusive trading by high-frequency traders (professional traders) though it could incorporate other algorithmic activity as well. There is a study [14] that introduces a new measure of low-latency activity to evaluate the influence of high-frequency trading on the market environment. The measure is highly correlated with National Association of Securities Dealers Automated Quotations (NASDAQ) constructed estimates of high-frequency trading, but it can be calculated from widely-available message data. They had used this measure to study how low-latency activity influences market quality both during customary market states and during a period of declining prices and heightened economic uncertainty. The analysis implies that increased low-latency activity improves traditional market quality measures - decreasing spreads, raising exposed depth in the limit order book and lowering short-term volatility. Given the contemporary market structure for U.S. eq-

uities, increased low-latency activity need not work to the detriment of long-term investors [14].

The financial environment is defined by an ever rising pace of both information gathering and the actions prompted by this information. Speed is important to traders in certain terms due to the natural volatility of financial securities. Related speed, in the sense of being faster than other traders, is also very significant because it can perform profit opportunities by allowing a prompt response to news or market activity. This last consideration appears to drive an arms race where traders apply cutting-edge technology and place computers in nearby proximity to the trading venue in order to decrease the latency of their orders and obtain an improvement. As a consequence, today's markets experience extreme movement in the "millisecond environment" where computer algorithms react to any other at a pace 100 times faster than it would take for a human trader to blink.

Exchanges markets have been investing heavily in upgrading their systems to overcome the time it needs to transmit information to customers, as well as to acquire and handle customers' orders. They have also begun to offer traders the ability to co-locate the traders' computer systems in proximity to theirs, thereby decreasing delivery times to under a millisecond. Since the traders spent technology resources to process information faster, the entire event/analysis/action cycle has been diminished to a couple of milliseconds.

There was a research that explores the impact of these low-latency traders on certain dimensions of market quality. More particularly, how their mixed activity affects attributes such as bid-ask spreads, the total price influence of trades, the short-term volatility of stocks and depth in the limit order book. To investigate these questions, they made use of publicly-available NASDAQ order-level information that is similar to those provided to supporters and provide real-time information about orders and executions on NASDAQ. Each entry (submission, cancellation, or execution) is time-stamped to the millisecond and thus, these data provide a very detailed view of NASDAQ activity. They mean that an expansion in low-latency activity reduces quoted spreads and the total price impact of trades, enhances depth in the limit order book and lowers short-term volatility. The results suggest that the increased activity of low-latency traders is beneficial to conventional benchmarks of market nature in the current U.S. equity market composition, one that is distinguished by both large fragmentation and wide usage of agency and proprietary algorithms [14].

The conclusion is that in the current market structure for equities, increased low-latency activity improves traditional standards of market quality such as liquidity and short-term volatility. It is of major importance that at times of falling prices and market anxiety, the nature of the millisecond environment and the positive influence of low-latency activity on market quality remains. Nevertheless, they cannot rule out the possibility of a sudden and severe market condition in which high-frequency traders contribute to a market failure. The occurrence of the "flash crash" in May of 2010 shows that such fragility is surely possible when a few big players step aside and nobody remains to post limit orders [14].

## 2.3 Backtesting

A fundamental distinction between a traditional investment management process and a quantitative investment process is the opportunity of backtesting a quantitative investment strategy to understand how it would have operated in the past. Even if a trader found a strategy described in complete detail with all the historical performance data available, he would still need to backtest it himself. This exercise serves several purposes. If nothing else, this replication of the research will ensure that he has learned the strategy effectively and has reproduced it exactly for implementation as a trading system. Just as in a medical scientific research, replicating others' results also guarantees that the original research did not perform any of the common errors plaguing this process. However, more than just performing due diligence, doing the backtest himself allows him to explore with variations of the initial strategy, thereby refining and improving the strategy. Many commercial platforms are designed for backtesting, some costing more than thousands of dollars.

### 2.3.1 Excel

Excel is the most basic and most common tool for traders, whether retail or institutional. They can improve its power further if they can write Visual Basic macros. The good of Excel is "What you see is what you get" (WYSIWYG). Data and program are all in one section so that nothing is covered. Also, a common backtesting trap called "look-ahead bias" is unlikely to occur in Excel (unless using macro) since is easy to align the dates with the different data columns and signals on a spreadsheet. Another benefit of Excel is that usually backtesting and live trade generation can be done from the same spreadsheet, reducing any duplication of programming works. The main disadvantage of Excel is that it can be used to backtest only fairly simple models.

### 2.3.2 MATLAB

Matrix Laboratory (MATLAB) is one of the most common backtesting platforms used by quantitative analysts and traders in large companies. It is perfect for testing strategies that involve a wide portfolio of stocks. Backtesting a strategy involving 1,500 symbols on Excel it is reasonable, but quite difficult. It has many excellent statistical and mathematical modules built in it, so traders do not need to reinvent the wheel if their trading algo's require some complex but basic mathematical concepts. There is also a huge number of third-party freeware available for download, several of them precious for quantitative trading purposes. Finally, MATLAB is very helpful in retrieving web pages with financial information and parsing it into a valuable form (so-called web scraping). Yet the seeming sophistication of the platform, it's truly extremely easy to learn and it is very fast to write a complete backtest program using this language. The main drawback of MATLAB is that it is comparatively expensive. However, there are many clones of MATLAB on the market where is possible to write and use codes that are very comparable to MATLAB like Octave, O-Matrix or Scilab.

These clones may cost hundreds of dollars or may be totally free. Not surprisingly, the more valuable the clone is, the fitter it is with programs written in MATLAB. The other disadvantage of MATLAB is that it is very useful for backtesting but not an appropriate execution platform. So, usually, it is necessary to build a distribute execution system in

another language once the algorithm has backtested the strategy. Despite these drawbacks, MATLAB has found extensive use in the quantitative trading community.

### 2.3.3 Trade Station

TradeStation is close to many retail traders as a brokerage that provides all-in-one backtesting and trade execution platforms connected to the brokerage's servers. The main advantages of this setup are:

- Largest of the historical data necessary for backtesting is readily available, whereas is necessary to download the data from somewhere else if using Excel or MATLAB.
- Once the program is backtested, it is possible to generate orders using the same program and send the orders to the brokerage servers.

As a limitation, the trader will be attached to TradeStation as the broker and the particular language used by TradeStation is not as adjustable as MATLAB and does not cover some of the more advanced statistical and mathematical functions many traders use. Although, if a trader prefers the ease of use of an all-in-one system, TradeStation may be a good choice [15].

### 2.3.4 MetaTrader 4

MT4 allows traders to test Expert Advisors before applying them in a live market. It allows traders to evaluate the Expert's efficiency and to prove that it works as expected. MT4's Tester is a multifunctional window (Fig. 2.1) where traders can experiment trading strategies and optimize an Expert's parameters to obtain the best combination of variables that will produce the most positive results. In the beginning, only the Settings and Journal tabs are noticed in the Tester window. Other tabs will appear as some actions are taken; for example, the Results tab appears next an Expert has been experimented.

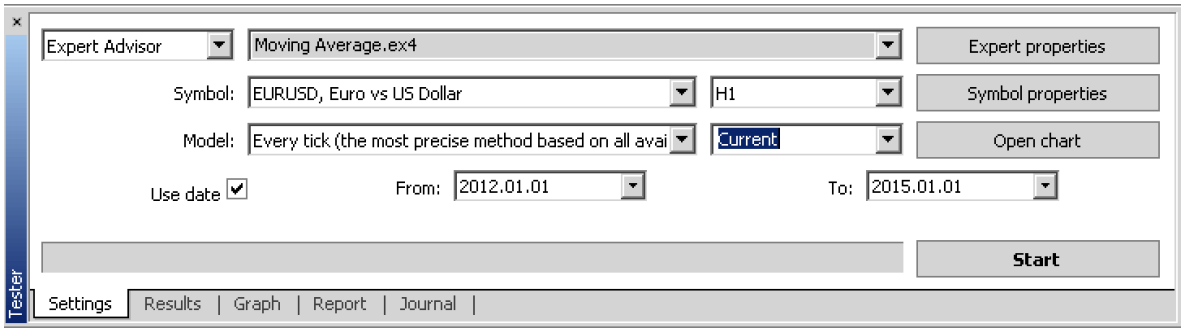


Figure 2.1: MetaTrader 4 Backtesting

The Tester window includes the following tabs:

- **Settings:** the settings of testing and optimization; e.g., the time interval to be experimented;
- **Results:** the results of the trade orders executed on past data;

- **Graph:** results displayed graphically;
- **Report:** a complete testing report;
- **Journal:** Expert's actions and internal messages log;
- **Optimization Results:** data about every optimization pass, covering inputs, profitability, and drawdowns;
- **Optimization Graph:** optimization results presented graphically;

To test an Expert Advisor, the trader will have to select some input parameters:

- **Expert Advisor:** Expert Advisors available for testing (Only those that are compiled);
- **Expert Properties:** Extended Expert Properties;
- **Symbol and Period:** The symbol is set in the Symbol field; the timeframe is defined in the "Period" field. If there is no historical data saved for the symbol or period, the Tester will download the last 512 historical bars;
- **Model:** One of three methods of historical data modeling can be chosen for testing:
  - **Open prices only:** the fastest method suitable for Expert Advisors that control bar opening;
  - **Control points:** results are considered estimates only;
  - **Every tick:** the most accurate method of modeling. Since this method involves a large amount of tick data, it is typically slow and can bog down the computer's operation.
- **Use Date:** The historical price data on which the test will be applied; complete the From and To fields to identify a range;
- **Optimization:** Check to enable the Expert parameters optimization mode; if it is disabled, the Expert will be tested but not optimized when the "Start" button is pressed;
- **Open Chart:** Opens a new price chart with the symbol selected for testing. The chart will show trade entries and exits, and can be opened only after the Expert has been tested;
- **Modify Expert:** This will open the MetaEditor to make changes in code if desired.
- **Start:** Start button will start the test/optimization.

MT4 can automatically create continuous approaches of the same Expert, with different inputs on the same data. Performing this optimization can help traders discover the inputs that have the most favorable results. To set up an optimization, traders must define which variables will be optimized by clicking on the "Expert properties". This will open a different window with three tabs:

- **Testing:** general optimization parameters (Fig. 2.2);

- **Inputs:** inputs are variables that affect the Expert's operation. Checked parameters will include inputs in the optimization; leave unchecked will disregard during optimization. If a parameter is checked, each field should be specified (Fig. 2.3);
- **Optimization:** the tab allows traders to use conditions during optimization. If any of the conditions is met during a separate pass of the optimization process, the optimization will be interrupted (Fig. 2.4);

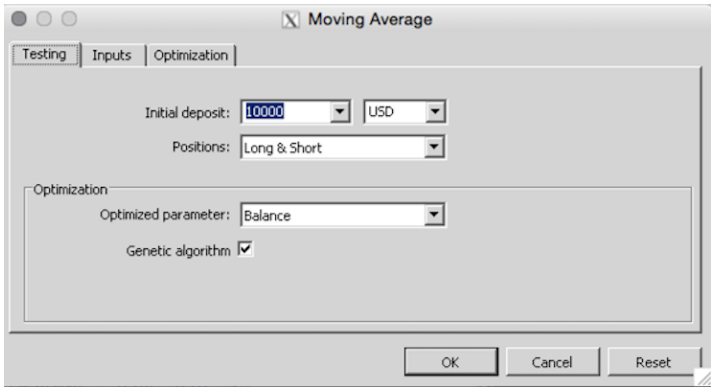


Figure 2.2: MetaTrader 4 Backtesting

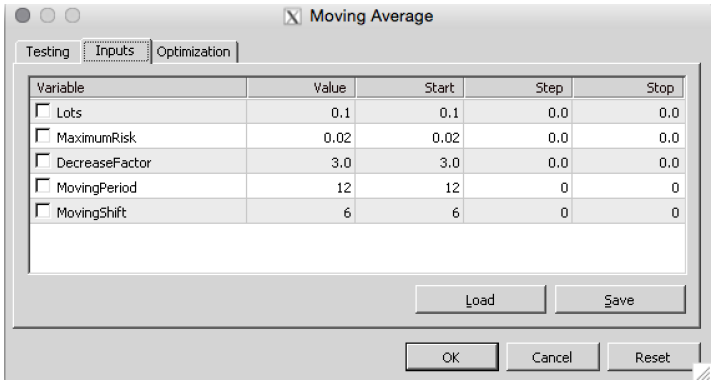


Figure 2.3: MetaTrader 4 Backtesting

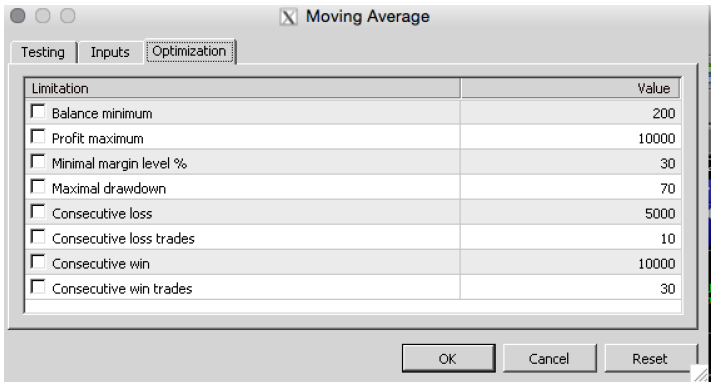


Figure 2.4: MetaTrader 4 Backtesting

Automated trading and strategy testing and optimization are advanced features of the MetaTrader 4 platform. Automated trading is popular because it excludes some of the risks from trading, helps traders bypass costly order-entry mistakes, and responds promptly to changing market conditions. The facility to test and optimize a trading idea (Expert Advisor) before putting it in a live market with real money is a valuable step in the development of a profitable trading system.

**Backtesting Summary**

Backtesting is about handling a historical simulation of the execution of a strategy. The belief is that the future performance of the strategy will match its past performance. There are several nuts and bolts involved in creating a historical backtest and in reducing the divergence of the future execution of the strategy from its backtest performance:

- **Data:** Split/dividend adjustments, noise in daily high/low and survivorship preference.
- **Performance measurement:** Annualized Sharpe ratio and highest drawdown.
- **Look-ahead bias:** Using unobtainable future data for past trading decisions.
- **Data-snooping bias:** Using several parameters to fit historical data and bypassing it using large enough sample, out-of-sample testing and sensitivity analysis.
- **Transaction cost:** Impact of transaction costs on performance.
- **Strategy refinement:** Common ways to make little variations on the strategy to optimize performance.

## 2.4 Algorithmic Trading Languages

### 2.4.1 Lua

Lua is an embeddable scripting language that points to simplicity, tiny size, portability and performance. Contrary to other scripting languages, Lua has an important focus on embeddability, supporting a development style where components of an application are written in a "hard" language such as C or C++ and parts are written in Lua. At present, Lua is used

in a wide range of applications, being seen as the principal scripting language in the game industry [16] [17].

Lua is a strong, fast, lightweight scripting language. Lua mixes simple procedural syntax with robust data description constructs based on associative arrays and extensible semantics. It is dynamically typed, runs over interpreting bytecode for a register-based virtual machine and has automated memory management by incremental garbage collection, making it perfect for configuration, scripting and rapid prototyping [17] [18].

Lua is applied in several industrial applications (e.g., Adobe's Photoshop Lightroom), with a weight of embedded systems (the Ginga middleware for digital TV in Brazil) and games (World of Warcraft and Angry Birds). Lua is, presently, the principal scripting language in games, it has a reliable reference manual and there are several books about it. Various versions have been released and are used in real applications since its creation in 1993 [17] [18].

Lua has a justified notoriety for performance. Several benchmarks show Lua as the fastest language in the area of interpreted scripting languages. Lua is fast not only in benchmark programs but *in vivo* too. Large fractions of large applications have been written in Lua. There is also LuaJIT for even better performance, an independent implementation of Lua using a just-in-time compiler [17] [18].

It is disposed in a small package and builds out-of-the-box on all platforms that hold a standard C compiler. Lua runs on all Unix and Windows, on mobile devices using Android, iOS, BREW, Symbian, Windows Phone, on embedded microprocessors, on IBM mainframes, among others [17] [18].

Lua is a fast language engine with a tiny footprint that is simpler to embed smoothly into an application. Lua has a simple and fully documented API that allows powerful combination with code written in other languages. Elongating Lua with libraries written in other languages is simple. It is also mild to extend programs written in other languages with Lua. Lua has been applied to extend programs written not exclusively in C and C++, but also in Java, C#, Fortran, Erlang and also in other scripting languages, such as Perl, Ruby or Python [17] [18].

A fundamental idea in the design of Lua is to provide meta-mechanisms for implementing features, instead of giving a host of features directly in the language. For instance, although Lua is not a true object-oriented language, it does present meta-mechanisms for implement classes and inheritance. Lua's meta-mechanisms cause an economy of concepts and keep the language small while allowing the semantics to be increased in unconventional ways [17] [18].

Combining Lua to an application does not bloat it. The tarball for Lua 5.3.2 that contains source and documentation uses 282K compressed and 1.1M uncompressed. The source includes approximately 24000 lines of C. Under 64-bit Linux, the Lua interpreter built with all standard Lua libraries takes 245K and the Lua library takes 419K [17] [18].

Lua is a free open-source software, shared under a liberal license (MIT license). It may be applied for any purpose, including commercial purposes, at absolutely no cost [17] [18].

Obviously, Lua is not the single scripting language around. Different languages can be employed for similar uses. However, Lua offers a collection of features that makes it the best choice for many tasks and gives it a unique profile.

### Lua Architecture

Although Lua offers a stand-alone command line interpreter, Lua is meant to be embedded in software. Applications can manage when a script is interpreted, loaded and executed. They can also catch errors, control multiple Lua contexts and extend Lua's abilities.

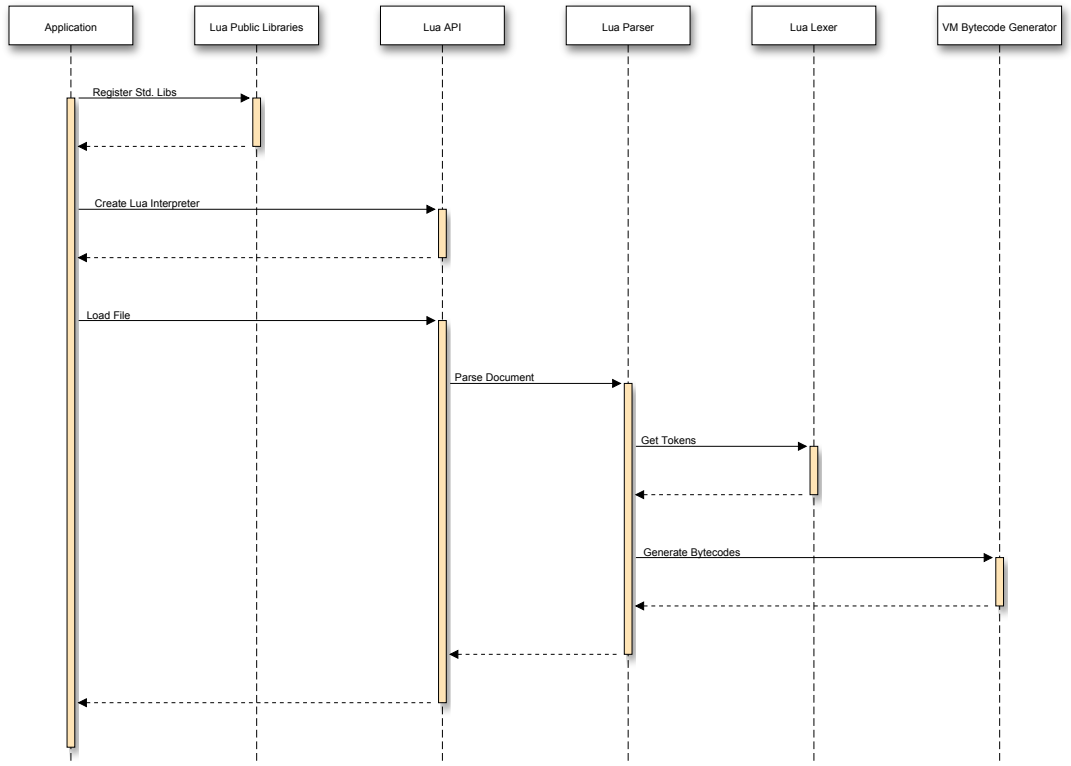


Figure 2.5: Process of initializing Lua and loading a script file

The method of initializing Lua and loading a script is described in figure 2.5. Four steps are required to load and execute a Lua script. To begin, a state of the Lua interpreter has to be created. This state is given on to every function of Lua's C API, comprising the calls done in the following steps. Second, the application embedding Lua records all libraries that extend Lua. Next, scripts given by the application are parsed and instructions that the virtual machine can execute are created. These instructions are applied as bytecodes. Ultimately, the bytecodes are sent to the virtual machine for execution.

Before loading and executing a script it is necessary to create a Lua interpreter reference. This action consists of initializing a "lua state" composition by calling "lua open". The "lua state" structure is needed because Lua offers a reentrant API and does not use any global

variable. As a consequence, an application may create various instances of the Lua interpreter.

Subsequently, the application must register libraries prepared to Lua programs. Lua holds a default set of libraries for end applications. Applications may extend or contact the list of libraries available to Lua programs by mastering which libraries are registered. This enables applications to customize the library functions possible to Lua applications.

Then, the Lua interpreter needs to get bytecodes to execute. At this moment, there are two potential scenarios: precompiled Lua bytecodes are loaded or a Lua script is loaded. While loading a script, Lua uses standard lexer, parser and bytecode generation modules to precompile the program. These components act like Pipes and Filters by transferring data to each other sequentially and incrementally. Because all of these components has a meaningful impact on performance, Lua needs to execute these elements as promptly as possible. In order to do that, Lua does not use automated code generation tools such as lex or yacc, alternatively, the language has a hand-written parser and lexer.

Finally, Lua needs to execute the bytecodes. The virtual machine kernel includes a loop that reads and executes a virtual machine instruction.

### 2.4.2 Python

Python programming language is especially adapted to quantitative analysts/programmers working in the area of financial engineering. This assertion centers on two axes: the first, Python is expressiveness and high-level quality; the second, Python's extensibility and interoperability with different programming languages. Other arguments are the advantages from the use of standard libraries and Python's support for functional programming properties [19].

Python is a general-purpose high-level programming language. Python's high-level quality and its rich collection of built-in data types enables the analyst/programmer to concentrate more on the problems they are working on and less on low-level mechanical constructs. The simplicity and notable expressiveness of the Python programming language syntax, gives productivity pickups that result from adopting Python above other languages. As a result of these features, programs in Python can be expected to be much tinier and more concise than their designs in other programming languages [19]. For quantitative analysts and computational scientists, several helpful Python packages exist to do the assignment of numerical analysis programs much simpler (e.g., SciPy). In addition, well know quantitative analysts in the field that are writing programs for finance will typically require much more than numerical code. The reason of this is because many of these programs are concerned with acquiring and organizing information on which the numerical aspects of the program are employed. Very often these tasks can be achieved in fewer lines of code and with significantly less effort in Python than other programming languages [19].

Python integrates well with data analysis, visualization and Graphical User Interface (GUI) toolkit. One more compelling argument for the adoption of Python is the efficiency with which Python integrates with visualization software such as GNUPlot2 making it possible for the analyst to construct personalized 'MATLAB-like' environments. Besides, they have neither the interest or time to invest in producing graphical user interfaces (GUIs). They can

be nonetheless important [19]. Python gives Tk-based4 GUI tools making it straightforward to wrap programs into GUIs [20].

Python plays well with others technologies. A diversity of methods exists to extend Python from the C and C++ programming languages. Conversely, a Python interpreter is simply embedded in C and C++ programs. In the world of financial engineering, C/C++ prevails and large bodies of this code exist in most financial institutions. The capacity for new programs to be written in Python that can interoperate with these code investments is a tremendous victory for the analyst and the institutions admitting its use [19].

### 2.4.3 MQL4

MetaQuotes Language 4 (MQL4) is a built-in language for developing trading strategies (2.6). This language is produced by MetaQuotes Software Corporation based on their vast background in the creation of online trading platforms. By using this language it is possible to create Expert Advisors that make trading management automated and are suitable for implementing trading strategies. Furthermore, using MQL4 is viable to develop technical indicators, scripts and libraries [21].

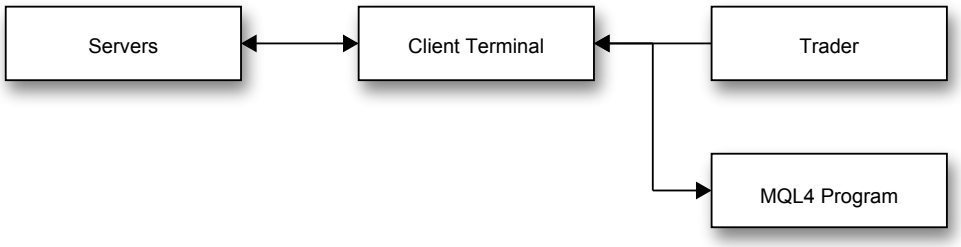


Figure 2.6: MQL4 Integrated Development Environment

MQL4 includes a large number of functions required for analysing current and earlier received quotes and has built-in basic indicators and functions for handling trade orders and managing them. The MetaEditor is a text editor which highlights diverse constructions of MQL4 language and is employed for writing the program code, figure 2.7 makes this clear to understand. It encourages users to orientate themselves in the expert system text quickly [21].

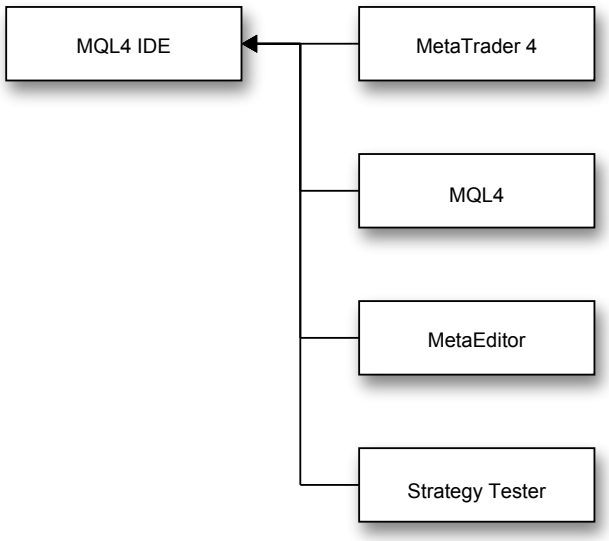


Figure 2.7: Automated Trading with MetaTrader 4

Programs written in MetaQuotes Language 4 have different features and purposes:

- **Expert Advisor** is a mechanical trading system connected up to a specific chart. An Expert Advisor begins to run when an event happens: events of initialization, an event of a fresh tick receipt, a timer event, depth of market-changing event, chart event and custom events. An Expert Advisor can both inform about a chance to trade and automatically trade on an account sending orders right to a trade server.
- **Custom Indicator** is a technical indicator written separately in addition to those already combined into client terminal. Like built-in indicators, they cannot trade instantly and are designed for performing of analytical functions alone.
- **Script** is a program intended for a particular execution of some actions. Unlike Expert Advisors, scripts do not process any actions, except for the start event.
- **Library** is a set of routine functions intended for storing and distributing commonly used blocks of custom programs. Libraries cannot start executing by themselves.
- **Include File** is a source of the regularly used blocks of custom programs. Such files can be entered into the source files of Expert Advisors, scripts, custom indicators and libraries at the compiling step. The use of included files is better than the use of libraries because of the extra load occurring at calling library functions [21].

## 2.5 Existing Solutions

### 2.5.1 MetaTrader

MetaTrader is an electronic trading platform extensively used by online retail foreign exchange traders. The first solution was developed by MetaQuotes Software and it was released in 2002. It is licensed to foreign exchange brokers who serve the software to their clients. The software has two parts, the client side and server side. The server component is run by a broker and the client software is provided to the broker's customers. The broker's customers are able to get online streaming prices and charts and to manage their accounts [22].

The client component is a Microsoft application with the capacity for end users to write their own trading script and robots to automate trading. In the client application, Metatrader supports many functionalities. It includes a built-in editor and compiler with access to a user contributed free library of software, articles and help. Metatrader uses a proprietary scripting language MQL5 that gives to traders the ability to develop expert advisors, custom indicators and scripts. One of the reasons for MetaTrader's popularity was its support of algorithmic trading [23].

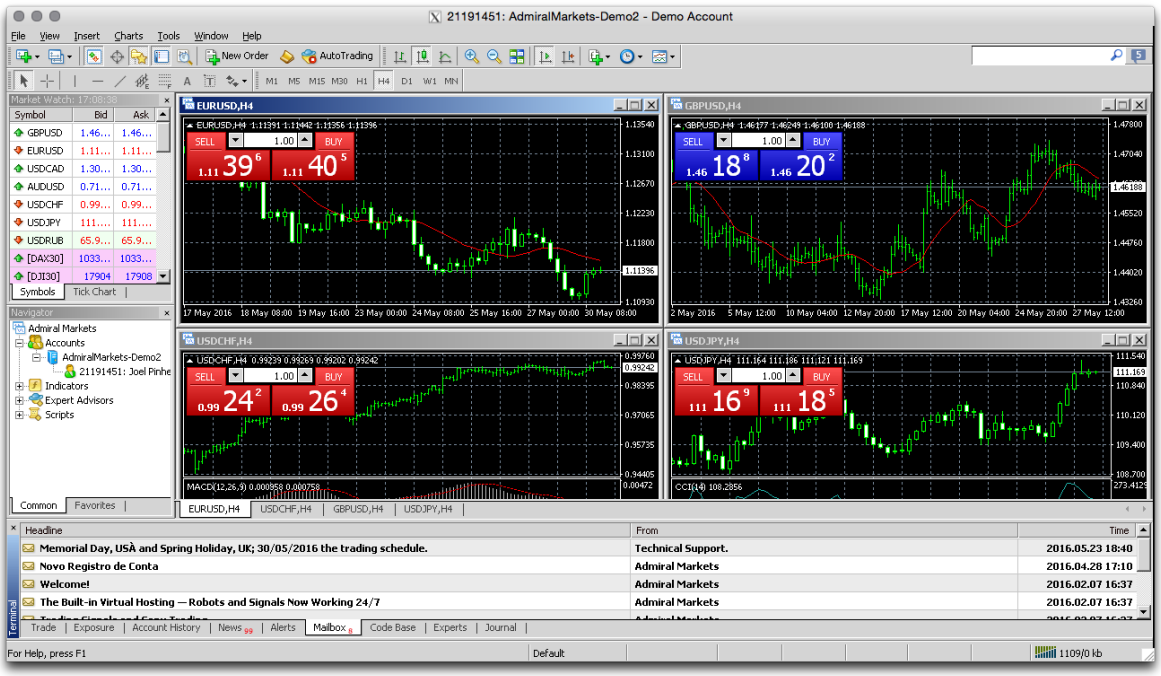


Figure 2.8: MetaTrader 4 Trading Platform Screenshot - OSX Emulated

MetaTrader is a stand-alone system (figure 2.8) where the broker is managing its position in the financial market. MetaTrader provides two ways of trading orders, Pending Orders and Market Orders. Pending Orders will be executed when the price reaches a predefined level. Market Orders can be accomplished in one of four modes:

- **Instant Execution:** the order will be executed at the price displayed in the client

terminal and at a known price. A good trading opportunity can be gone when the volatility is high and the requested price cannot be served.

- **Request Execution:** this approach enables the trader to executed a Market order in two steps. First, a price quote is requested, then, a trader decides whether to buy or sell using the received price. A trader has several seconds to decide if the received price is a benefits trading. In this mode, it offers a certain knowledge of price combined with guaranteed execution at that price. The tradeoff is the reduced speed of execution, which can take a lot longer than other modes [24].
- **Market Execution** the orders will be executed with the broker's price no matter if it is different from the one displayed in the client terminal. The disadvantage of this mode is that deviation can get considerable during volatile price changes.
- **Exchange Execution:** the order is processed by an external execution facility, the exchange. The trade is executed according to the current depth of market [25].

MetaTrader package includes the Client Terminal, Server, Administrator and Data Center (Figure 2.9). The client terminal is provided freely by brokerages for real-time online trading. This affords trade operations, charts and technical analysis in real-time. There's also a trading script language that allows users to program trading strategies, indicators and signals. This software runs only on Windows platforms. Hacks were made to use MetaTrader in Linux and Mac-OS (OSX), but they are all using Windows emulators. The Server is designed to handle user requests to perform trade operations, display and execution of warrants. It also sends price quotes and news broadcasts, records and maintains archives. MetaTrader Administrator is designed to remotely manage the server settings. There's also the Data Center which is a specialized proxy server. It can be also an intermediary between the server and client terminals, reducing the price quote sending load on the main server.

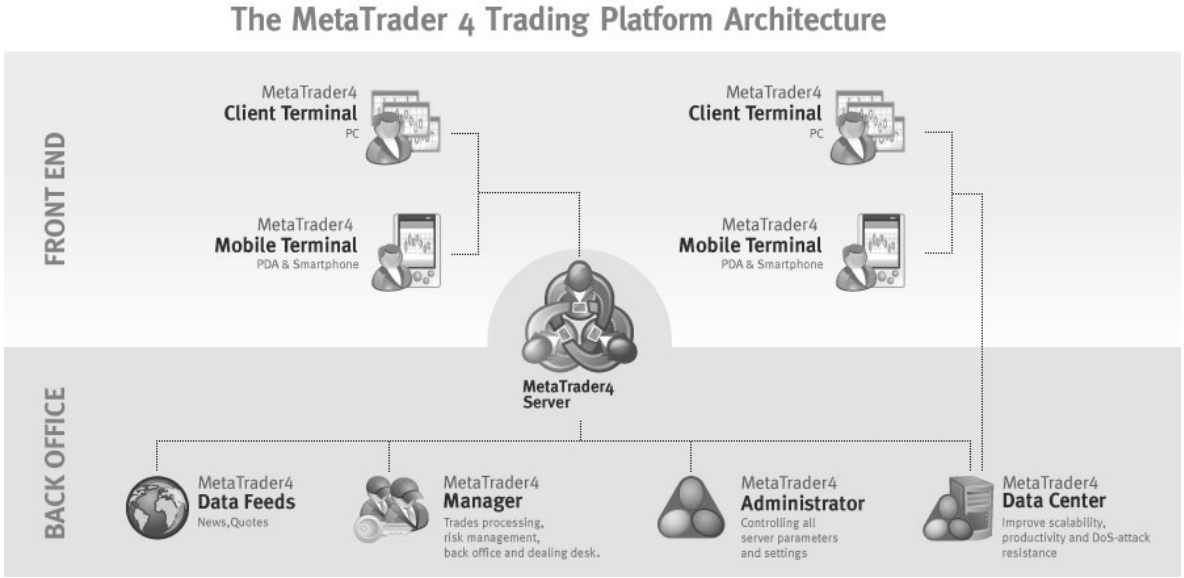


Figure 2.9: MetaTrader 4 Trading Platform - Architecture
[26]

#### Portability

As said before, one of the problems with MetaTrader is its portability. MetaTrader supports natively only Windows platform. For other platforms, hacks were made to use MetaTrader. In Linux some traders are using MT4 through Wine [27], in OSX through WineBottler [28]. Besides the performance, certain features may not work (graphical issues) and crashes could happen. OSX Clients of Metatrader are truly unhappy with the current OSX/Linux versions. Theirs disappointment is clear in the MetaTrader website [29], who claim a native version, not a virtualized one.

#### Performance

Since OSX and Linux platforms need an emulated environment to run MetaTrader its performance will be aggravated naturally. For instance, tools like Wine will introduce an extra layer above the system, which will make the system to run slowly. Using a different configuration it is possible to improve the MetaTrader performance.

#### Graphics

The interaction with MetaTrader plots is not ideal and there aren't many types of plots to choose. This is one of the negatives points in MetaTrader. There are many applications of the same type that give 10 types of different graphics and plots.

#### Features

MetaTrader is a complete trading platform. Its features are the trading system itself, analytics functions, trading signals, MetaTrader Market (where traders can find an expert advisor or a technical indicator), algorithmic trading, backtesting, mobile trading and alerts/financial news. These features makes MetaTrader the most complete and used trading platform.

#### Technologies

This trading solution was written for .NET platform. Because of that, there are a lot of issues in running MetaTrader in a non-native system. MetaTrader should have a native version for Windows, Linux and Mac or, at least, a non-virtualized one.

Summing-up, MetaTrader is the most popular financial trading platform of the market. It has great functionalities such as trading scripting language, analysis features and good graphics. However, it only supports Windows's platform. Platforms like Linux or OSX have to run by some Windows emulator with nonofficial configurations. By consequence, running emulated MetaTrader performance becomes poor.

### 2.5.2 Protrader

#### Protrader for Desktop

Protrader desktop is an all-in-one trading software that couples expert tools with a completely customizable interface (figure 2.11). It has a high-level functionality and opens access to trade multiple exchanges with low latency. The large list of features are multi-asset trading,

market analysis, developing and running of algorithmic strategies, risk management, among others [30].

Protrader interface permits the trader to customize every individual panel settings as well as the whole workspace completely. Any user can effortlessly design a trading layout according to special requirements or preferences using a wide range of alternatives. For instance traders can use docking panels on various monitors to make a more personalized workspace.

Protrader desktop software (figure 2.10) allows trading on different asset classes including Forex, Stocks, CFDs, Options and Futures. Instant access to multiple markets allows multi-asset trading profits. The broker can continually expand the number of traders without the necessity to modify the trading platform.

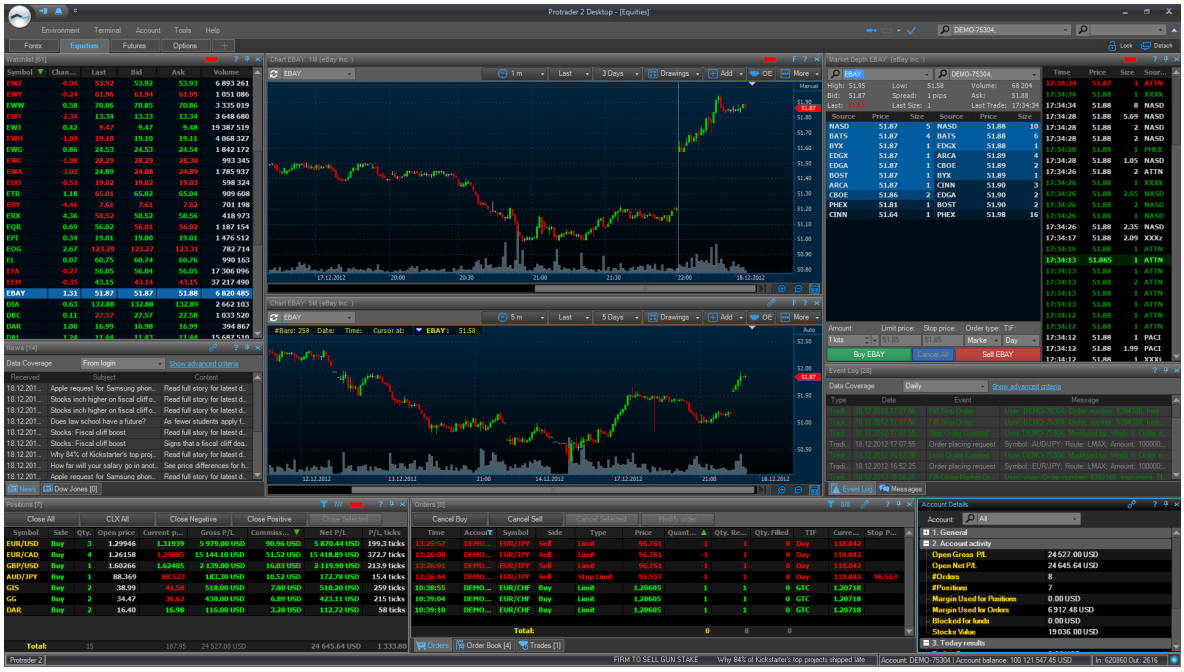

Figure 2.10: ProTrader Screenshot

Charts view provides clients with general and high-level chart types, entirely customizable time frames and toolbar, overlay function and valuable set of user settings. Chart trading increases speed due to position management right on the chart as well as embedded order entry panel enhances the convenience of the trading process.

Programming and testing of algorithmic strategies are available using AlgoStudio functionality since it supports C# and MQL4 languages. It includes strategies scripting, debugging, optimization and backtesting. Walk-forward optimization, strategy examination and real-time statistic give traders a deeper analysis of the strategy.

Protrader gives to traders many great professional tools:

- **Scalper:** An expert panel for short-term high-frequency trading and it gives compre-

hensive functionality: Level 2 quotes, Time & Sales data, tick charts overlaying and mouse trading mode (e.g., to place a limit order it is as simple as a click on the price: left click for buy, right click for sell.);

- **Saved Orders:** A simultaneous opening of the positions by basket parts from many markets became practicable due to Saved Orders panel. Custom sets of orders can be saved and send on demand;
- **Visual Trading:** Protrader chart trading instruments raises the pace of managing multiple positions and orders. Drag and drop orders to change, close and open positions just by one click;
- **Market Depth:** Tracking market viewpoint and order flow become viable via professional trading tool Market Depth. It presents Level 1 and Level 2 quotes with a built-in order listing;
- **Time & Sales:** An expert analytic tool, which displays information about trades and level 1 quotes data. It is crucial for tape reading of current market information;
- **Option Master:** Permits measuring option positions, plotting option profiles, showing their variation depending on different market circumstances and testing the existing option positions;
- **Matrix:** This module was created to cooperate with Market Depth. The Matrix provides the capability for a single click order insertion, modification and cancellation;
- **Portfolio:** This mechanism is helpful for those who trade portfolio strategies. It is plausible to build a graph of the portfolio, set coefficient of instruments' values and trade portfolio as a single asset.

**Protrader Web**

Protrader Web is a light and easy-to-use software, based on HyperText Markup Language (HTML). Traders can work it on every modern web-browser and operating system including Mac-OS or Linux. Protrader Web interface is widely customizable whereas the list of trading tools can serve any trader from beginner to expert. It allows usage of exceptional chart functionality, a large-scale modern trading instruments and workspace customization [31].

The principal feature of Protrader for Web application is being a cross Operating Systems software. Presently, traders don't have to install any software on their PCs to begin trading, they simply launch the personal browser, log into Web station using their own accounts and start trading.

Protrader panels can be tagged in groups for space saving and connected by a symbol for immediate instrument switching. Many workspaces allow creating an easy trading environment for several trading styles.

The principal benefit of Web application is that it is sufficient for most traders. Current web technologies allowed to implement the entire central trading functions that are available in Protrader desktop version. The list of characteristics includes advanced charting functionality,

Level 2 quotes, Time & Sales panel and layout panels linking.

Protrader offers great professional tools:

- **Market Depth:** Implementation in web application increases standard functionality and gives Level 1 and Level 2 quotes;
- **Times & Sales:** Concludes Market depth information by tape trades and Level 1 quotes through expanding resources of the web station;
- **Visual Trading:** Best Protrader desktop trading characteristics find their implementation in this Web-based trading platform. Visual trading increases capabilities of front-end as one of the principal trading tools for all applications;
- **Indicators:** Each trader can examine price action using specific tools like indicators. Protrader Web-based trading software has more than 50 indicators that will be allies in market viewpoint estimation;

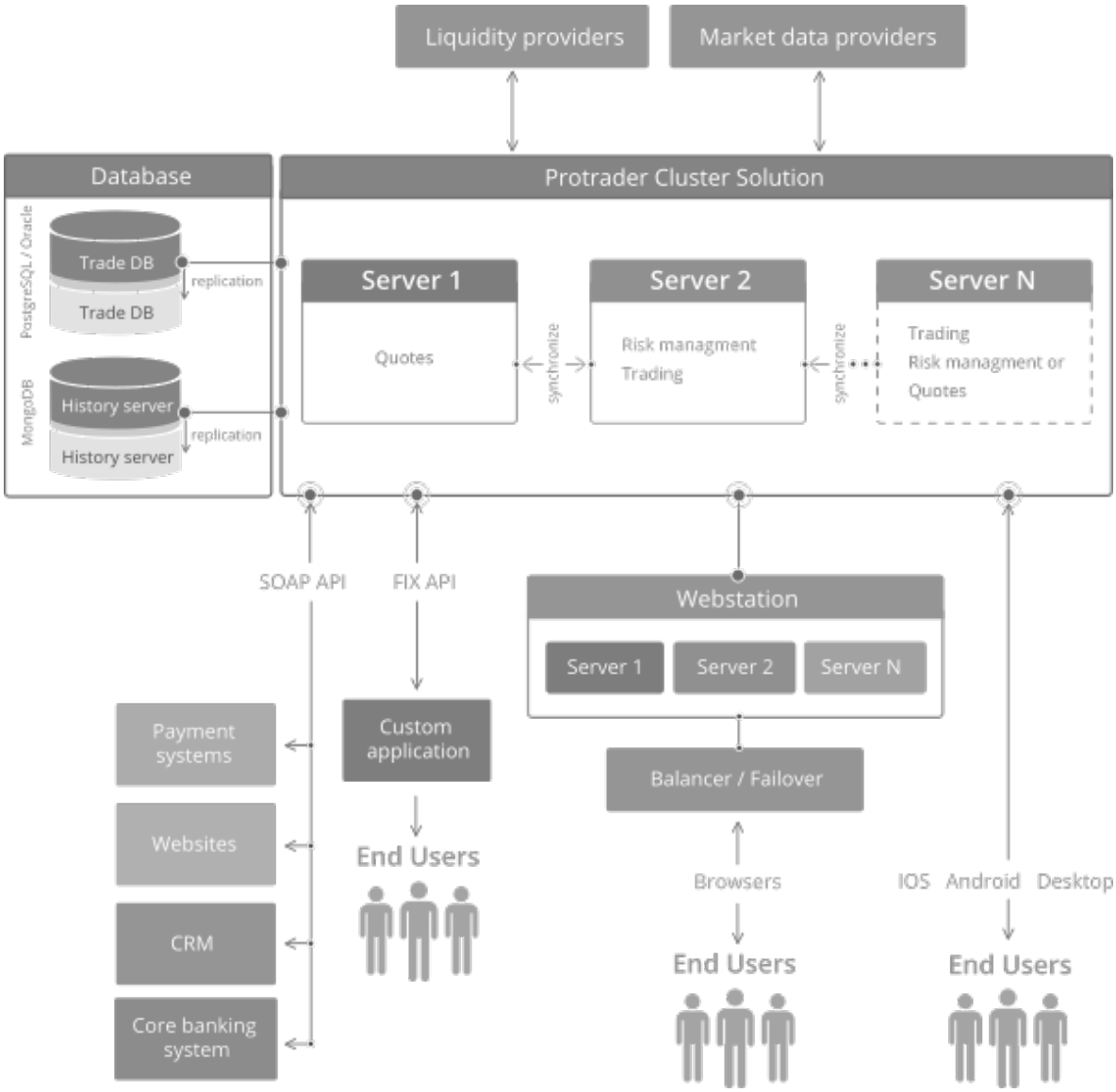


Figure 2.11: Protrader - Server Architecture
[32]

**Portability**

One obstacle that Protrader has is that it is exclusive to Windows platforms. Protrader had unique supported .NET platform. The Lead .NET developer, Alexey Bogdan said that "The most used Personal Computer (PC) platform - is Microsoft Windows, that is why our software can be suitable for the most users" [30]. This shows their plans in the subject of native Protrader versions. Recently, they had created Protrader for The Web that is a basic package of the popular Protrader for Desktop features. Since then, Protrader can to run in a non-windows platforms such as OSX and Linux. However, the official affirmation "Protrader is a Cross-OS application" [31] isn't really true, indeed. In fact, Protrader for The Web isn't a portable version of Protrader for Desktop, it is actually a basic version of Protrader.

#### Performance

Protrader for Desktop performance is acceptable since it is developed to be a native application. Regarding the Web version of Protrader, once it is developed under cutting-edge web technologies its GUI is extraordinarily lightweight but has some aesthetic problems.

#### Graphics

Graphics capabilities of Protrader aren't great for both versions. Trading applications should be richer in charts, plots and with tools embedded in graphs.

#### Features

Protrader for Desktop has a lot of features as cited above. However, its cross-OS application is very limited. Protrader for the Web doesn't have an algorithmic programming platform embedded in the application. It is a big issue for OSX and Linux users who want to develop their scripts using AT.

#### Technologies

Protrader native application was written for .NET platform. Protrader developed a cross-OS application that runs in all OSs using HTML5, Cascading Style Sheets (CSS) and JavaScript).

### 2.5.3 Plus500

Plus500 is a corporation which provides online trading services to retail clients (figure 2.12). The corporation was founded in 2008 and provides trading in CFDs (contract for difference) on a variety of financial markets. The company's central operations are based in Israel, with controlled entities in the United Kingdom (UK), Cyprus and Australia. Shares of the parent company, Plus 500 Ltd, are listed on the Alternative Investment Market of the London Stock Exchange. Plus500 is Europe's No. 1 CFD provider by a plenty of new traders.

Plus500 group had developed a Windows Application natively. A Web Version was created for OSX and Linux users. The big problem for OSX and Linux is that it doesn't have all features of Windows Application. All the characteristics of a native application such as notifications, docker app, tray symbol, access to file system and others, are not allowed in a Web Application. Its native Windows application performance is good. However the Web Application is inferior because of the natural web latency. Plus500 only offers two types of graphics in its Web Application for OSX and Linux users. It is clear insufficient for the most traders users who want a long list of tool to analyse their trades. Plus500 Web version is a minimal version of Windows application. Because of that, this web software has much fewer features. All OSX and Linux users can't get the same of Plus500 as Windows ones.

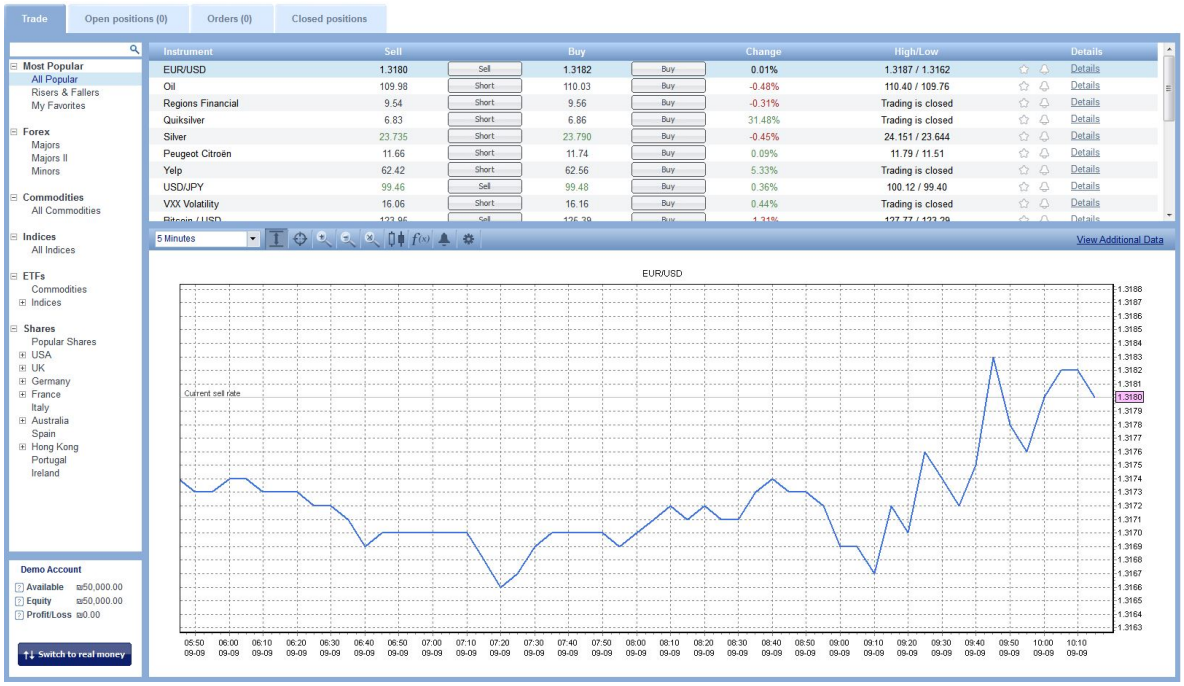


Figure 2.12: Plus500 Screenshot

### 2.5.4 MetaStock Trader

MetaStock is a proprietary trading software originally released by Computer Asset Management in 1985. It is one of the most popular stock trading software applications, offers more than 300 technical indicators, built-in drawing tools like Fibonacci retracement to complement technical indicators, integrated news, fundamental data with screening and filtering criteria, with global markets coverage across multiple assets. It has both real-time and end-of-day versions [33].

### 2.5.5 Worden TC2000

Worden TC2000 is a great option for those who are interested in US and Canadian stocks and funds. Their specialties include charts, watch lists, alerts, instant messaging, news, scanning and sorting. It offers basic data coverage, more than 70 technical indicators with 10 drawing tools, an easy-to-use trading interface and backtesting capacity on historical data. However, there is not support for automated trading tools [34].

### 2.5.6 eSignal

eSignal is a stock trading software that gives analysis capacities. It has global coverage within various classes of assets including stocks, funds, bonds, derivates and forex. eSignal is valuable on trade management interface with a news span. Existent technical indicators appear to be limited in number. It comes with backtesting and alert specialties [35].

### 2.5.7 NinjaTrading

NinjaTrader is a combined trading software providing an end-to-end solution from order entry to execution with customized development options and third-party integration (more than 300 add-ons). It is one of the frequently used trading and analysis platforms. Aside from the common technical indicators, fundamentals, charting and analysis means, it also supplies a valuable trade simulator, allowing risk-free trade learning for young traders [36].

### 2.5.8 Wave59 Pro

Giving excellent products for expert traders, Wave59 provides high-end functionality, comprising "hive technology artificial intelligence module, market astrophysics, system testing, integrated order execution, pattern building and matching, the Fibonacci vortex, a full suite of Gann-based tools, training mode and neural networks" [37].

### 2.5.9 EquityFeed Workstation

An example of an excellently highlighted characteristic of the EquityFeed Workstation is a stock hunting tool called "FilterBuilder" - built in a large number of filtering criteria that empowers traders to scan and choose stocks per their wanted parameter. Level 2 market data is also possible and coverage includes OTC and PinkSheet markets. Still, it offers insufficient technical indicators and no backtesting or automated trading [38].

### 2.5.10 ProfitSource Platform

Aimed to busy traders for regular trading, ProfitSource Platform alleges to have an edge with complex technical indicators, particularly Elliot Wave examination and backtesting functionality with more than 40 automated technical indicators built-in. Its asset class covers equities, options, futures and funds [39].

### 2.5.11 VectorVest

VectorVest has trading and analytics software platforms. It gives a full coverage for traditional technical indicators beyond major stocks and funds. VectorVest also contributes with strong backtesting capabilities, customization, real-time filtering, watch lists and charting tools [40].

### 2.5.12 INO MarketClub

For users particularly looking for charting tools, INO's MarketClub offers technical indicators, trend lines, quantitative analysis tools and filtering functionality mixed with a charting and trading solution [41].

### 2.5.13 Existing Solutions Summary

Many software solutions are available from brokerage companies and independent vendors declaring diverse functions to support traders. The important part of choosing the best product should be based on the output functionality that best suits trading requirements. Amateur traders should choose software applications which have a solid reputation in the market. It expected basic functionality at a low cost. Veteran traders can examine individual products selectively to match their more precise criteria. The follow table 2.1 can resume the solutions based on tree criteria features: Backtesting, Algorithmic Trading and Indicators.

| Solution | Backtesting | Algotrading | Indicators |
|---|---|---|---|
| MetaTrader | ✓ | ✓ | ✓ |
| ProTrader | ✓ | ✓ | ✓ |
| Plus500 | ✗ | ✗ | ✗ |
| MetaStock | ✗ | ✗ | ✓ |
| Worden TC2000 | ✓ | ✗ | ✓ |
| eSignal | ✓ | ✗ | ✓ |
| NinjaTrading | ✗ | ✗ | ✓ |
| Wave59 | ✓ | ✗ | ✗ |
| EquityFeed | ✗ | ✗ | ✗ |
| ProfitSource | ✗ | ✗ | ✓ |
| VectorVest | ✓ | ✗ | ✓ |
| VectorVest | ✗ | ✗ | ✓ |

Table 2.1: Summary of Solutions

## 2.6 Financial Brokers APIs

Nowadays, the biggest brokerage firms offer free or premium trading software applications to singular clients when they initiate a brokerage account. These software applications, giving a deep diversity of trade, examination and analysis functions, are used as a leading sales-pitch to the trader client. They also own features as in-built technical indicators, fundamental analysis numbers, integrated applications for trade automation, news and alert features [42] .

By definition, a broker is "an agent who buys or sells for a principal on a commission basis without having title to the property". In another way, "a person who functions as an intermediary between two or more parties in negotiating agreements, bargains, or the like" [43].

In investment world a 'broker' is:

- An individual or a firm that charges a fee or commission for executing buy and sell orders submitted by an investor;
- The role of a firm when it acts as an agent for a customer and charges the customer a commission for its services;

- A licensed real estate professional who typically represents the seller of a property. A broker's duties may include determining market values, advertising properties for sale, showing properties to prospective buyers and advertising clients with regard to offers and related matters[44].

Commonly, only the rich could manage a broker and access the stock exchange. The Internet cause an explosion of discount brokers, which allow investors to buy at a moderate cost, but doesn't supply personalized advice. A discount broker is a stockbroker who carries out buy and sell orders at a reduced commission compared to a full-service broker but provides no investment advice. For those who want to do their own experimentation or don't want to spend a lot of money, a discount broker is an attractive way to invest.

There is also the "Deep Discount Broker" who is an intermediary that mediates sales and exchanges between securities buyers and sellers at even lower charge commission rates than those given by a normal discount broker. As a single might expect, deep discount brokers also contribute to fewer services to clients comparing to standard brokers. As mentioned before, these brokers only provide the fulfillment of stock and option trades, pricing a flat fee for each.

It's not conceivable to start investing without a brokerage account. As a newborn investor, selecting the perfect broker is usually very different than it would be for older investors. Choosing a broker isn't all that distinct from choosing a stock since it requires careful reflection and not all brokers are fit for all investors.

There are some points that an investor should examine before picking a broker. There are two kinds of brokers: those who trade straight with their clients (regular brokers); and those who operate as intermediaries between the client and a larger broker (broker-resellers). Regular brokers generally are considered more notable than broker-resellers.

Trade execution taxes are essential, but there are other brokerage fees to judge. If an investor is under 30, there are a lot of chances that he has limited funds. When it comes to invest in this age, looking at prices that might apply is vital to ensure that he is making the most of his investment. There are some extra costs to weigh:

- **Minimums:** Most brokers have minimum balances for opening a brokerage account. Generally, this estimate ranges between $500 and $1,000 with an online discount broker.
- **Margin:** While new investors might not aspire to start a margin account right away, it's something to think about eventually. Margin accounts usually have raised minimum balance requirements than standard brokerage accounts.
- **Withdrawal:** Sometimes it's arduous to take out money from broker to the account. This is because brokers sometimes impose fees to make a withdrawal.
- **Complicated Fee Structures:** While most brokers have comparable fees schedules, some brokers have complicated fee structures that make it harder to sort out hidden fees.
- **Type of investor:** The decision of the right broker should be determined by the type of investor. No single broker is serves for all investors. Defining the investment style before starting is commonly a wise decision.

- **Trader:** Traders don't hold stocks for a long time. They are more engaged in quick and dirty gains based on short-term price volatility and they make many trade executions over a short time span. So, a trader will want to look for a broker with very cheap execution fees, as high trading fees could instantly eat up the returns.
- **The Buy-and-Hold Investor:** a Buy-and-Hold investor, or passive investor, is somebody who holds stocks for the long term. They're interested in making the value of their positions appreciated over longer periods of time and repeating the gains at a later date. If this is the investor type it must avoid brokers with monthly fees. For a long-term investor, a higher trade commission should be less of a concern.

It's absolutely essential to balance the requirements as an investor and as a client. While the first broker won't surely be permanent, an investor has a much greater chance of making money if he puts the right amount of time and research choosing a broker [45].

### 2.6.1 Interactive Brokers API

Interactive Broker [46] provides several APIs which a developer can use to link to his system and trade with an Interactive Brokers (IB) account. The API grants a connection through either the Trader Workstation (TWS) or the IB Gateway. Connectivity through the TWS requires the application running, but also allows to test and confirm that the API orders are working correctly. Connecting through the IB Gateway enables the use of API without a large GUI application running, but does not provide an interface to test and confirm API activity.


Figure 2.13: IB TWS Trades sent from iTrading

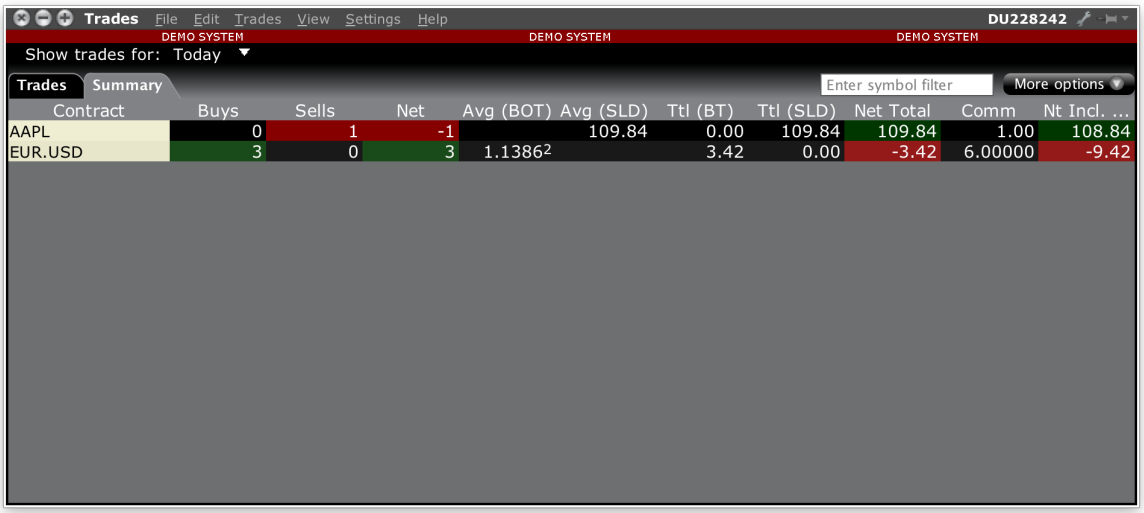


Figure 2.14: IB TWS Summary

The IB Gateway provides a low-resource alternative to TWS for connecting to the IB Trading system via API. The gateway uses approximately 40% fewer system resources than TWS. However, the gateway is GUI-less, which means that an individual cannot view the API activity as you can when running TWS.

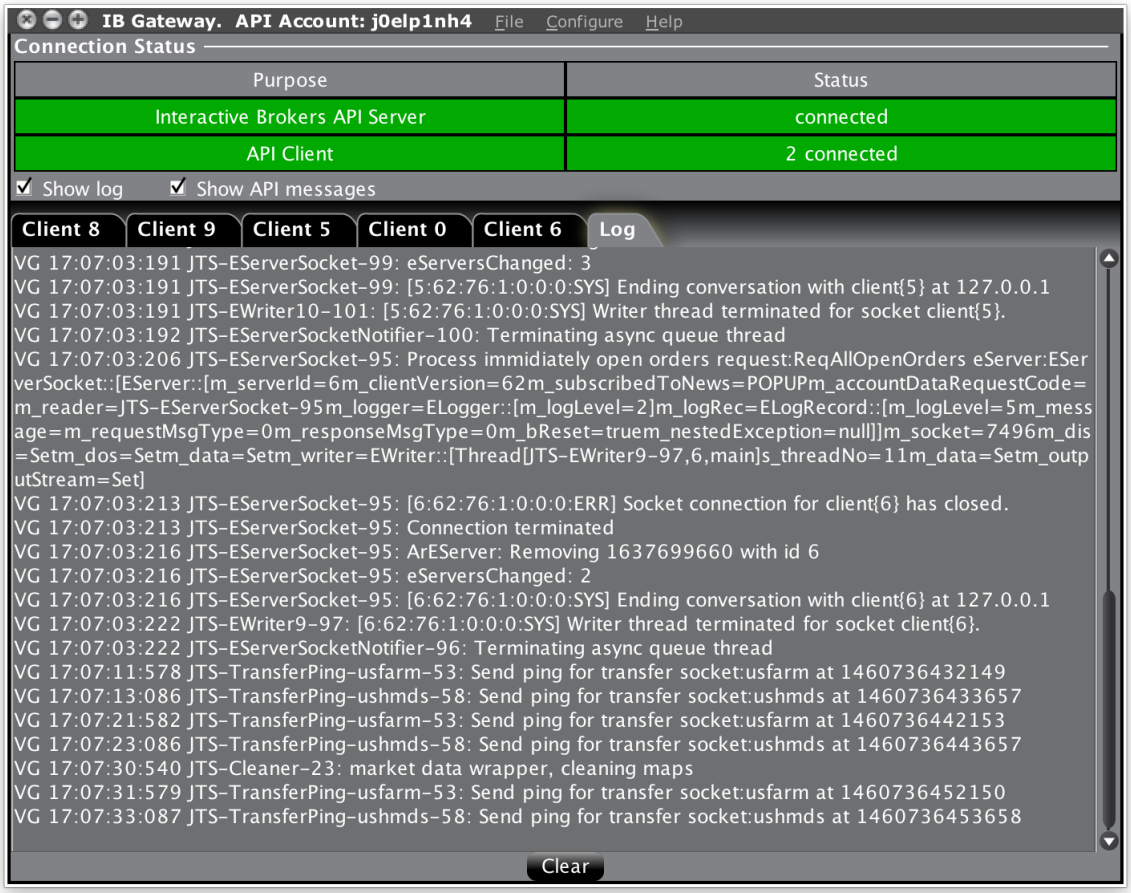


Figure 2.15: IB Gateway Log

In both IB TWS and Gateway version there is an option to select if it is Read-Only Mode. The read-only mode provides display-level data but does not allow requests for which trading access is required.

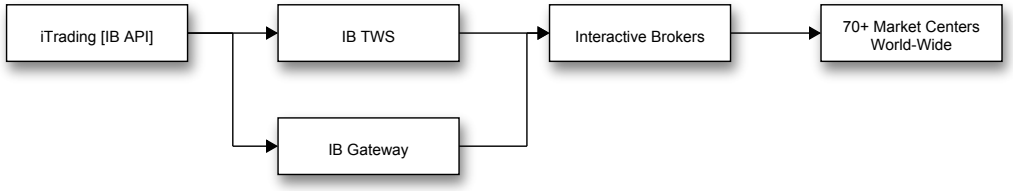


Figure 2.16: iTrading-Interactive Brokers interaction Diagram

### 2.6.2 MB Trading Broker

MB Trading offers a full range of services to integrate an application in a scalable, secure, and easy-to-use set of web-driven set of API's. It provides a real-time trading environment to

clients for FOREX, Futures, Equities and Options. MB Trading made the trading API as easy as possible. No specialized formats or message types. They use simple POST parameters, designed for speedy development and distribution. Besides, they offer distributed data to clients. Real-time and historic display data APIs, from streaming data to chart bars down to 1-min, all available through a standard HTTP-based API. Their services include track balances, holdings, and recent trades across single and multiple accounts simultaneously. Preview, submit, review and modify orders in real-time with direct-access speed. Easy to read documentation with a full staff of development support professional and open developer network. Full trading development can often be completed in less than thirty days by a single developer.

### 2.6.3 Chapter Summary

In this section it was discussed how algorithmic trading can have a huge impact on financial markets and why trading solutions should have into consideration low-latency orders. Some tools of backtesting were discussed and their behavior. After that, three possible algorithmic trading languages were explained with its advantages and disadvantages. Then, it was made an analysis to several trading applications and their respective features. Finally, it were analysed two broker companies.

# 3 Proposed Solution

Chapter 3 presents the proposed solution. Several technologies were analysed, in order to find out which one best fits it. Afterwards, it is described the functional requirements and wireframes of the system. Finally, it is detailed the architecture of the solution and the mindmap reasoning to achieve it.

## 3.1 Technologies and Frameworks

Business needs, functional requirements and speed of development sometimes requires cross-platform desktop applications. In this section it will be outlined the advantages and drawbacks of the most common technologies for these types of applications. The biggest players on the market are Java, QT and JavaScript-based solutions such as Chromium Embedded Framework (CEF) and Electron.JS. These solutions are popular worldwide and they have large developer communities. Many real, working and successful projects are based on them such as code editors, team collaboration tools, among others.

### 3.1.1 JAVA

One of the oldest approaches to cross-platform development is Java. Its slogan can describe it perfectly: “write once, run anywhere”. Java is one of the oldest platforms which means it has a big community, a lot of third-party libraries that solve rich variety of tasks such as data processing. Besides, it supports object-oriented programming model and multithreading. The performance is a point in favor since code optimization relies massively in system-specific features. However, Java Virtual Machine needs to be installed on the computer that runs the application. The support for Graphical User Interface (GUI) remains weak. Java is a technology to consider if there are requirements for libraries for data processing, multithreading or security. Java is a programming language expressly intended for use in the distributed environment. It was designed to have the "look and feel" of the C++ language, but it is easier to use than C++ and supports a pure object-oriented programming model.

#### Performance/Portability

Java Virtual Machine (JVM) is an environment that executes Java programs. Java programs are compiled into a neutral language called bytecode, which is what JVM executes.

Thus, any program compiled into bytecode can run on any platform that has a JVM installed on it. In short, Java is compatible with several different computing platforms.

Java programs that work on a Java Virtual Machine tend to perform slower than similar programs written in C++. The system neutrality of bytecode is a weakness in performance terms. Code optimization relies massively on system-specific features. Since Java bytecode is system-neutral, it cannot be optimized for a particular hardware set.

#### Graphics

Beyond the instability problem, Java's graphics model has plenty of features that makes it difficult to understand. From a graphic center perspective of using Java, the most problematical features are the following:

- "Forgetful bitmaps" and the lack of an automatic double buffering mechanism;
- The relative statelessness of the graphics context;
- The use of a resolution-dependent, pixel-based, non-Cartesian coordinate system.

Java's GUI libraries are adequate to create financial charts. There are few tools to create them and they are all limited in features [47].

### 3.1.2 C++/Qt

Qt is a cross-platform application framework that is broadly used for developing application software. It can be driven on different software and hardware platforms with small or no change in the underlying codebase. On the other hand, it is being used as a native application due to its abilities and speed (see Qt SDK figure 3.1). Qt is being developed both by the Qt Company and the Qt Project under open-source governance, including individual developers and firms working to improve it. This project is available with both commercial and open source licenses. [48]

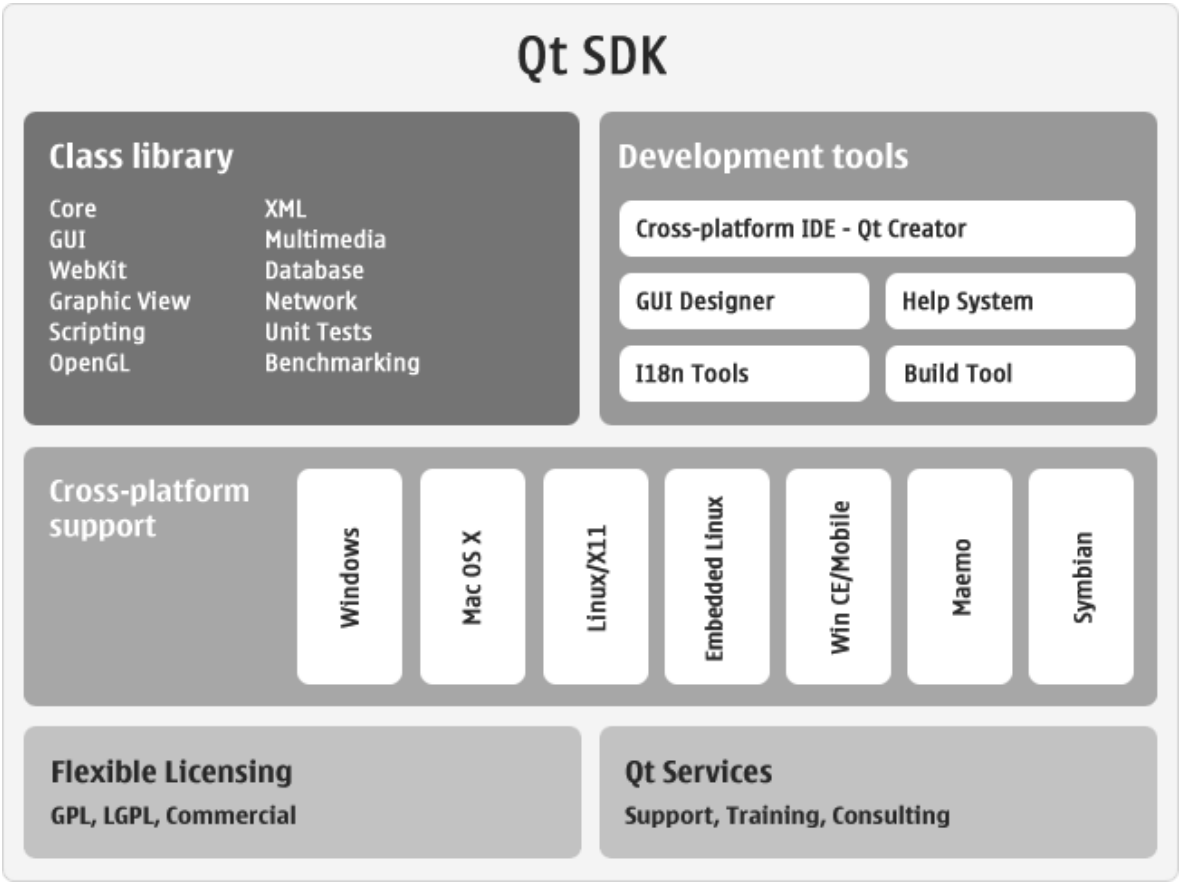


Figure 3.1: Qt SDK Diagram
[49]

Qt is applied mainly for developing software with graphical interfaces (GUIs), though programs without GUI can be developed, such as command-line tools and consoles for servers. GUI programs created with Qt can have a native-looking interface and in such cases, Qt is classified as a widget toolkit.

### Performance

Running native API is the best option, anything other than that is a wrapper around native API. Qt offers multithreading support so that a developer can have responsive GUI in one thread and whatever else in other threads without much trouble.

### Portability

Qt is a cross-platfrom application framework that is used for developing software. It is very versatile due to the lack of changes in the underlying codebase to develop applications on different software/hardware platforms.

### Graphics

Since Qt is written in C++ there are few tools to create charts and plots to Qt. The best is QCustomPlot[50], but is very expensive. To develop an application with great charts and plots Qt is not the best option.


Figure 3.2: Qt experimentation

### 3.1.3 Chromium Embedded Framework (CEF)

The Chromium Embedded Framework is an open source project that allows developers to quickly display HTML content in their desktop applications. The HTML view can be finely commanded and even extended by the available API. Below, the HTML rendering is done through the Chromium browser project, on top of the Blink engine (formerly WebKit) and the V8 JavaScript virtual machine [51] [52].

As frameworks normally go, Chromium Embedded Framework (CEF) affords essentially a set of headers and a library. It is available on Windows, Mac OSX and Linux. C++ an C APIs are free as part of the project, but there are projects out there which have wrappers for other environments like .NET, Java and Python.

CEF appears in two versions - CEF1 and CEF3, both of which are actively supported. These two projects present almost the very same API to embedding clients while rendering HTML using the Chromium flavor of WebKit (now Blink). Where they diverge is in how they catch into the HTML engine underneath, but to better get that, it is useful to have a comprehensive overview of the Chromium architecture.

The Chromium architecture has three main layers: Blink (formerly WebKit) API, Content API and Chrome. The Blink API gives access to the rendering and V8 JavaScript engine which commonly run in a single process. The Content API adds the multi-process architecture and provides implementations for more difficult HTML5 and browser features like accelerated compositing, Geolocation and WebRTC. The Chrome layer covers the Chrome browser user interface and feature implementations like history management, extensions and bookmarks that are tightly coupled to the Chrome browser. The Chromium organization is currently working to introduce a fourth concept, called Components, which will give modular implementations for browser features that span many layers.

The Blink and Content APIs are not well-built and several features have extra implementation requirements. CEF provides these implementations along with a stable API that hides most of the underlying Chromium and Blink complexity. CEF1 uses the Blink API directly and, as a result shares the single-process architecture. CEF3 uses the Content API and advantages from the multi-process architecture and many of the advanced features that are implemented in the Content API. As an added benefit for CEF3 the Components changes will make it easier to selectively enable and share feature implementations that at this time cannot be shared due to tight coupling with the Chrome layer.

#### Performance

The performance of CEF is compared with the quality of the JavaScript (JS) frameworks used. However, JS doesn't have good benchmarks when compared to Java or C++ [53] [54].

#### Portability

QT allows to launch software on the majority of modern operating systems by simply compiling the application for each Operating System (OS) without modifying its source code. It has one UI library for all platforms, supports multithreading and uses C++ as the programming language. Nonetheless, it requires to know platform-specific UI libraries and the use of third-party libraries is limited. The size of the application will be considerable, since all the libraries used in the application are included. Additionally, the distribution of applications for Windows are complex because of its license. It is easy to find issues with plugins and it is hard to find examples for non-standard cases. Finally, QT's popularity is much lower than Java, thus QT should be considered if a specific library is already implemented on C++ and there are no analogs for other platforms or languages.

#### Graphics

With HTML5 and an advanced web browser support, JavaScript has turned into the tool of choice for building high-performance web graphics [55]. The main reason for that is D3.JS. It is a JavaScript library created to display digital data in a dynamic graphical form. It helps to deliver data to life using HTML, Scalable Vector Graphics (SVG) and CSS. D3 enables excellent control over the definitive visual outcome and it is the cutting-edge and most convincing web-based data visualization technology on the market today [56].

### 3.1.4 Electron Atom Shell

Electron is an open source framework written by Cheng Zhao, an engineer who serves GitHub in Beijing on the Atom text editor team. Atom text editor combines Chromium and Node into a single runtime proper for building custom desktop web applications that also have access to Node for tasks that web browsers usually can't do [57].

Earlier, before starting Electron, Cheng Zhao participated massively to the node-webkit (called now nw.JS) project. Electron is conceptually similar to nw.JS but has some important technical distinctions. A key distinction is that Electron works with Google Chromium Content Module [58] to bring in Chromium functionality vs NW.JS which uses a forked version of Chromium itself [59].

Electron is base level. The Electron API, comparable to Node, is designed to support a rich userland of modules and applications. For example, there is a module called menubar that hides much of the complexity of the Electron API from the developer and lets him make a 'menubar' style app (e.g. Dropbox) in just a several lines of code. So, the developer can wrap the Electron API in a higher level module (just like Node.JS).

Apps work cross-platform. Node itself has supported Mac, Windows and Linux uniformly since version 0.6. Chromium is also cross platform. The Electron API philosophy is that it only adds support for features that can run on all platforms. For example, Windows has a 'system tray' but Mac Os has a 'menubar'. Electron implements an abstraction over these called the 'Tray' API that is generic enough to function on whatever platform it is running on. Electron itself doesn't include a way to package your code into an executable (e.g. an .app for Mac or an .exe for Windows), so there is a module designated electron-packager that allows the developer to build Mac, Windows or Linux apps from the very source code.

There are some important organizations using Electron in addition to GitHub. Notably Microsoft with their VisualStudio Code editor [60] and Facebook with their Nuclide editor [61]. Still, Electron can be used for many things beyond Code editors. For example, there is an app called ScreenCat [62] that is a screen/keyboard/mouse sharing plus voice chat app that works with WebRTC. If somebody has ScreenCat running it is possible to share the screen with someone else who has ScreenCat, or share the screen with someone in a WebRTC enabled web browser. It's a great tool to do remote pair programming with coworkers. Another example is Friends [63], it is a highly experimental decentralized public chat app, similar to Slack (which was implemented in Electron too) or Internet Relay Chat (IRC) but build uniquely on WebRTC Peer-To-Peer systems so it doesn't depend on central server allowing all communications to be exchanged directly between users.

Consequently, Electron is a growing open source project largely maintained by a single individual. Browsers are complex and, given the interest it has had so far, Electron will grow into a strong open source project, with many core contributions and even better cross-platform support.

Common web pages are designated to be executed inside a web browser with high control and restricted access to OS resources such as file system due to security ends. That's why

it is impossible to build a web application that is communicating with native systems and resources. Electron framework provides the possibility to create a desktop application with access to system resources using popular web technologies such as HTML, CSS and JavaScript. The framework is based on JavaScript runtime io.JS [64] (Node.JS fork with greater support of new features) and web browser Chromium [65] (open-source of Google Chrome).

#### Performance

The performance of Electron is related with the quality of the JS frameworks used. However, JS doesn't have good benchmarks when compared with Java or C++ [53] [54] However, using Node.JS it is possible to develop non-blocking applications leading to scalable and robust applications.

#### Portability

Developing and maintaining native desktop applications is very complex and many companies are pushing users towards web or cross-platform versions. There have been a plethora of options for accomplishing this over the decades. Flash, Air, Java and Silverlight are all options that promised this capability with varying degrees of success. The main problem with these options is that they generally involved learning another language or forced users to install plugins plagued with stability, performance and security problems. The power of JavaScript and Web technologies is well known and have seen a wave of options for developing and packaging cross-platform desktop apps using this knowledge. Electron, by GitHub, is one option, that has empowering many popular applications. Electron has recently reached version 1.0, which is always a milestone in any project's existence.

#### Graphics

Similarly to section 3.1.3, JavaScript is a great tool of choice for building web graphics because of D3.JS [56].

### 3.1.5 Technologies Summary

There are several reasons why JavaScript can be a great choice for desktop development. It is currently the most popular and widespread programming language. It is extremely hard to find more powerful and convenient layouting and stylization tools than HTML and CSS. There is an enormous variety of JavaScript libraries for different cases, which can be used in desktop applications. Besides, a big chunk of the application code can be used on other platforms: on Web as a site and on mobile platforms (phonegap).

All things considered, the technology used in this work was Electron.JS. Several reasons were taken into consideration. There are a lot of available libraries in Web to create what iTrading needs such as Lua VM, Integrated Development Environment (IDE) look-and-feel, interactive charts, bi-directional communication, documentation helpers, among others. Lately, Electron has been the most contributed repository from those based on Node.JS.

## 3.2 Functional Requirements

### 3.2.1 Use Cases

The use-case model illustrated in figure 3.3 is a representation of how diverse types of users interact with the global system to solve a problem. As such, it defines the goals of the users, the interactions among the users and the system and the expected behavior of the system in accomplishing these ends. A use-case model consists of some model parts. The most relevant model elements are use cases, actors (entities who interact with the system) and the relationships among them. This use-case design serves as a unifying thread throughout system development. It is the first specification of the functional requirements of the system. It allows to analyse and design, interaction input planning, determination of test cases and user documentation.

There are two actors, the Trader and the Broker (Interactive Brokers). All of them provide a unique and valuable perspective of the system. The functionality of it is set by different use cases, all of which represent a specific goal, to obtain the result [66].

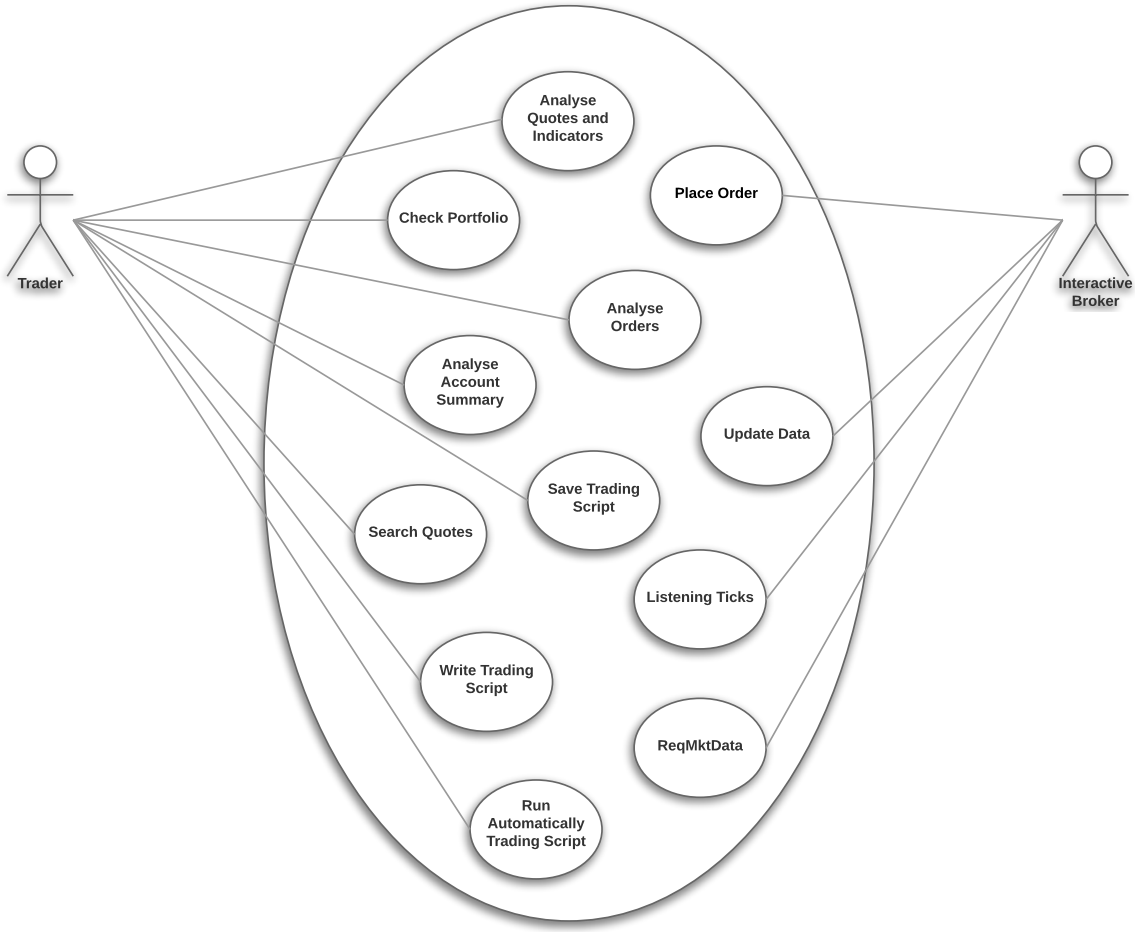


Figure 3.3: iTrading - Use Case Diagram

Table 3.1: Requirements: Use Cases

| Use Case | Description |
|---|---|
| *Analyse Quotes and Indicators* | Ability to have a graphic-centric screen where a trader can visualize and analyse financial markets using many graphic options. |
| *Place Order* | Place an order on the brokerage company using a broker API. |
| *Check Portfolio* | Capacity to observe the trader positions. |
| *Analyse Orders* | Area where a trader can analyse all the orders that were emitted to the broker. |
| *Analyse Account Summary* | The screen on which the solution user can look in detail to his positions to see his gains and losses. |
| *Update Data* | Action made by the broker that is constantly updating its data to the client. |
| *Save Trading Script* | Ability where the trader can, after writing his script, save and decide if he wants to put it running in automating mode. |
| *Search Quotes* | Search for data to analyse. |
| *Listening Ticks* | Broker is always feeding with all the changes that occur in a certain product. This use case will enable trades to run automatically. |
| *Write Trading Script* | System should have an IDE where a trader could write its scripts. |
| *RegMktData* | Financial broker will feed the client application with market data. |
| *Backtesting* | After the execution of the trade, trader can backtest his algorithm using historical data. |
| *Run Automatically Trading Script* | Run all the scripts defined to run "Automatically". |

## 3.3 Wireframes

A wireframe is a two-dimensional representation of an interface that explicitly focus on space allocation and prioritization of content, available functionalities and expected behaviors. For these purposes, wireframes typically do not include any styling, color or graphics.

It were made some wireframes during the designing of iTrading prototype (figure 3.4). These wireframes serve many purposes:

- Connect the information to its visual perspective by revealing paths between screens;
- Define consonant ways for representing particular types of information on the user interface;

- Determine expected functionality in the interface;
- For a given item, prioritize content by the determination of an adequate space as well as where it should be located.

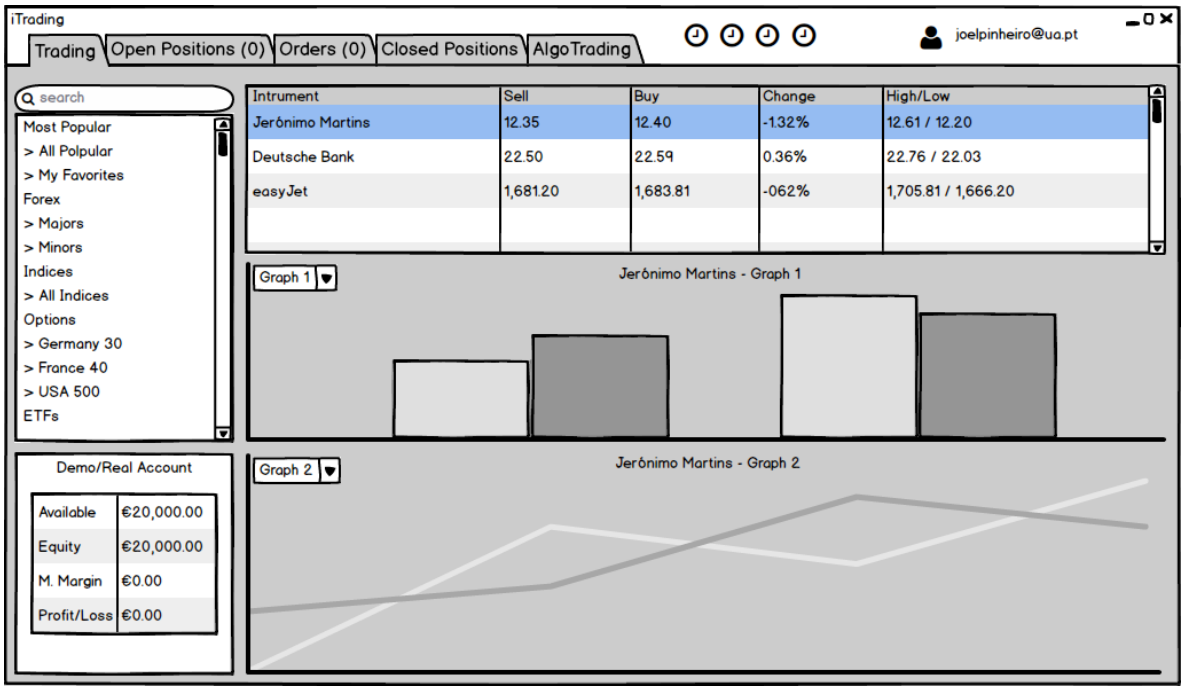


Figure 3.4: A Wireframe of iTrading - Trading Window

Figure 3.4 illustrates the interface of the very first screen of iTrading application. This screen serves as an analysis tool that a trader has in order to search the most popular financial products. It has several types of graphics, where the trader can choose to its preferences the one he feels more adequate to his needs.

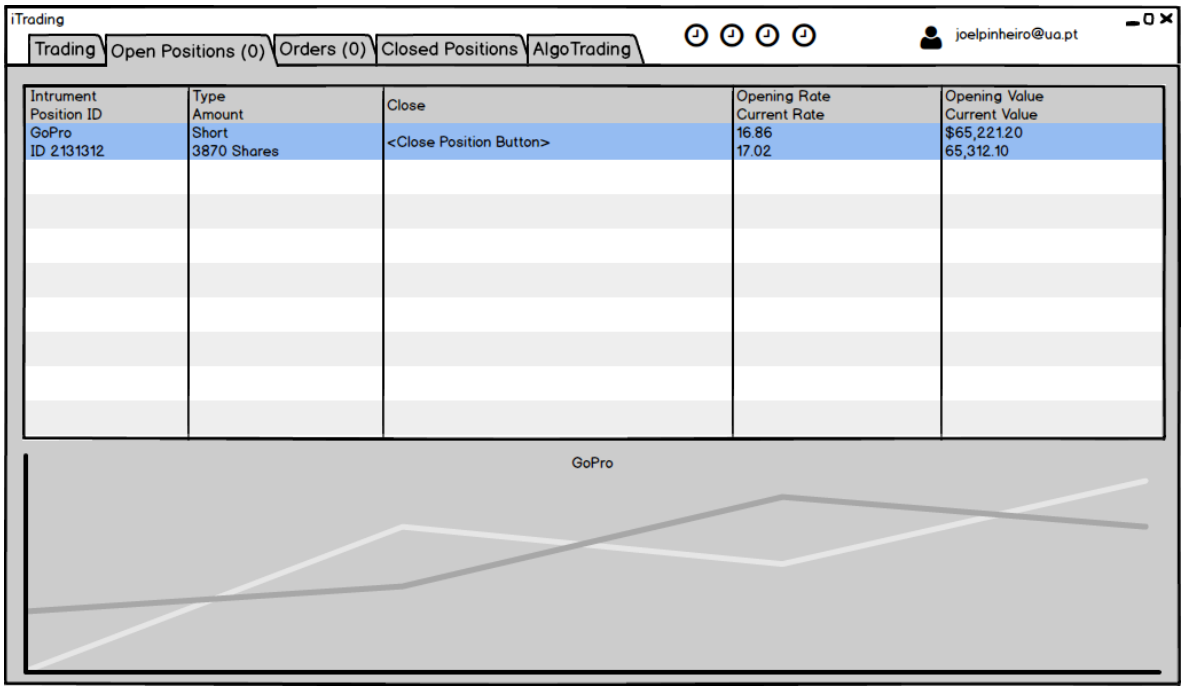


Figure 3.5: A Wireframe of iTrading - Open Positions Window

There are three screens related to positions: Open Positions, Orders and Closed Positions. Figure 3.5 shows the open positions which are the ones the trader holds.

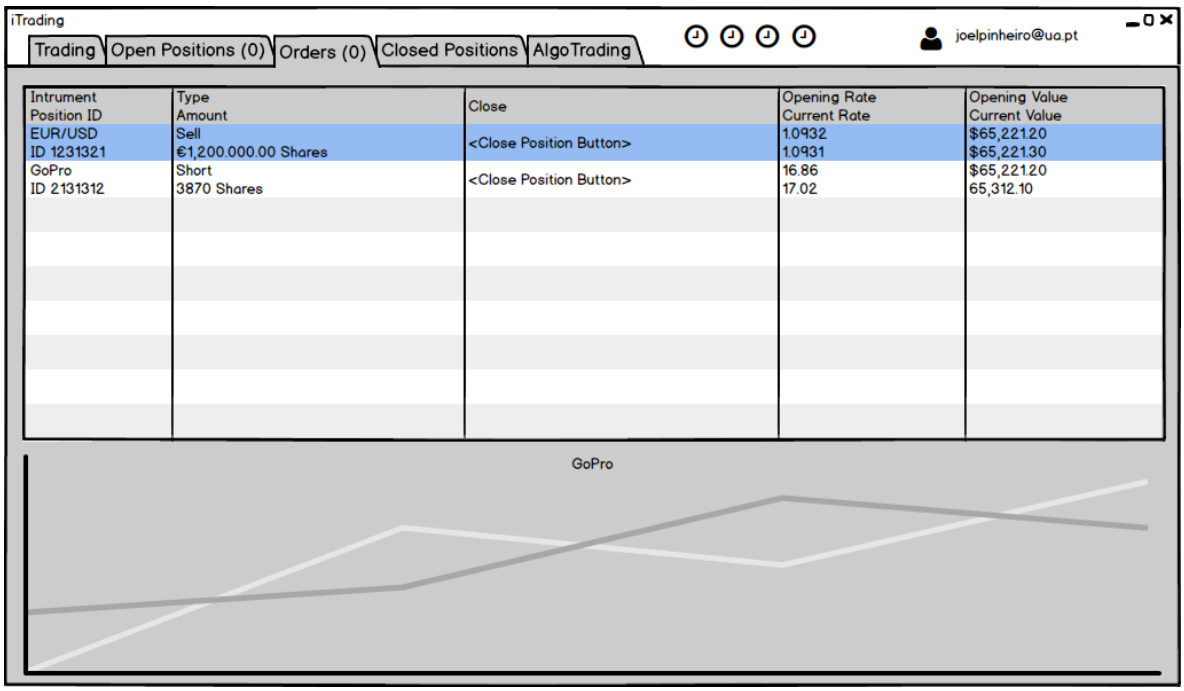


Figure 3.6: A Wireframe of iTrading - Orders Window

Figure 3.6 shows the total orders that were bought or sold by the trader.

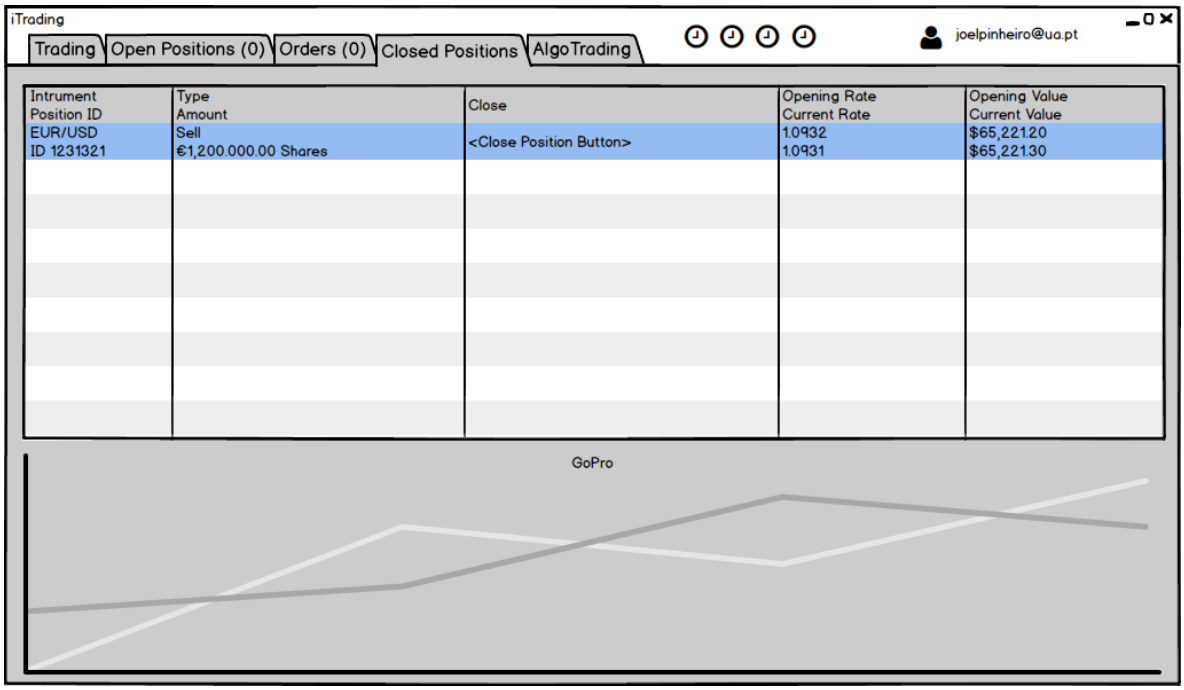


Figure 3.7: A Wireframe of iTrading - Closed Positions Window

Figure 3.7 illustrates the positions that were sold.

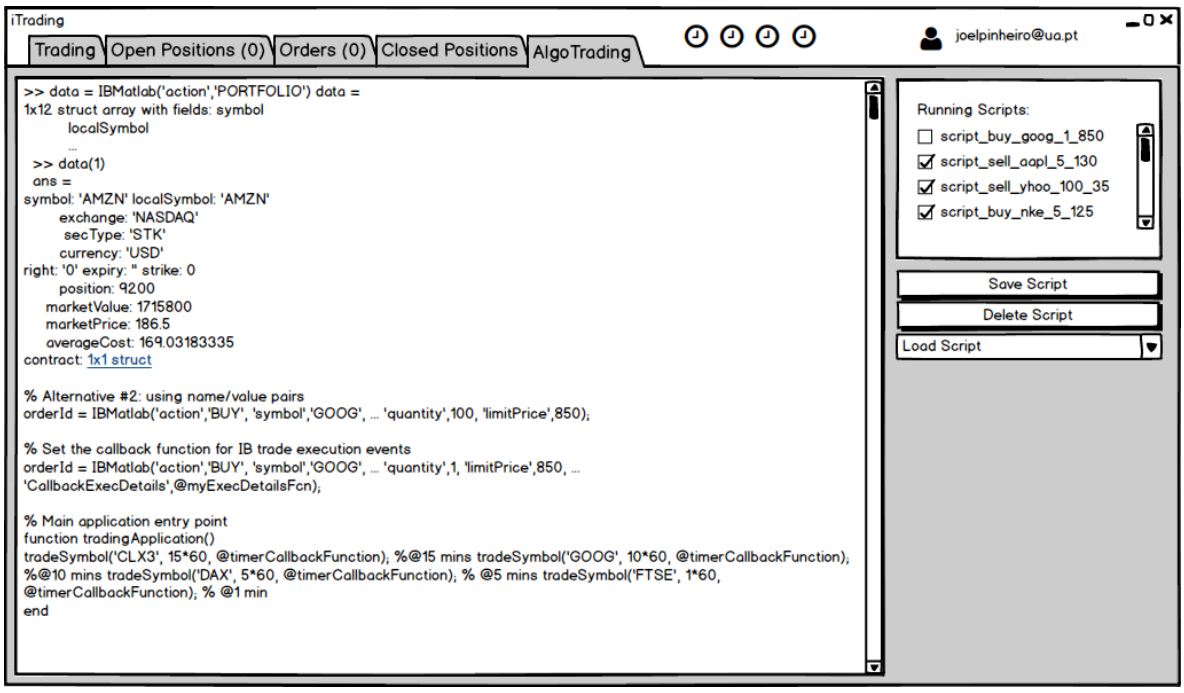


Figure 3.8: A Wireframe of iTrading - Algo Trading Window

Figure 3.8 is where the algorithmic trading, trading automation and backtesting is. In this screen a trader has the possibility to write his scripts. He can backtest, save and automate

them.

## 3.4 Architectural Proposal

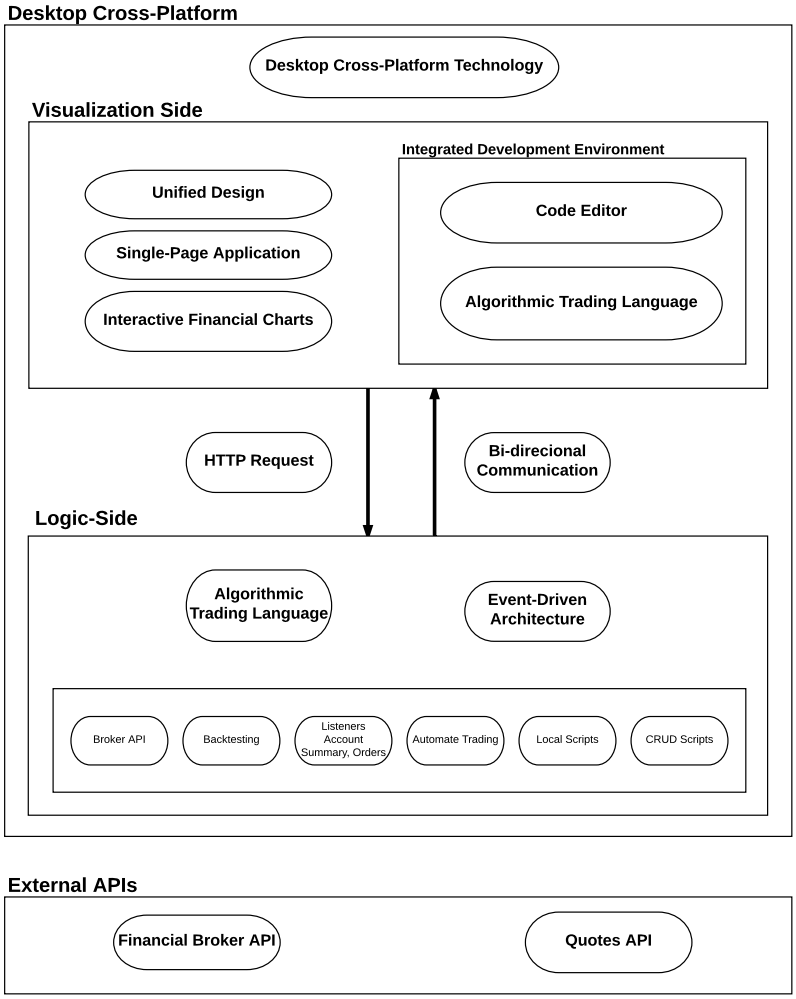


Figure 3.9: Proposed Architecture

Figure 3.9 shows the architecture of the proposed solution. The architecture can be decomposed into two domains. A desktop cross-platform and its Externals APIs. First, the desktop cross-platform should be developed using an adequate technology in order to enable the execution of the software in all platforms: Windows, Linux and OSX. It should be used at least two types of external APIs: the financial broker and a quotes API. The desktop cross-platform application is divided into two parts: visualization side and logic side. In the visualization side it must exist an Integrated Development Environment with an appropriate Domain-Specific Language (DSL) to develop algorithmic trading. Besides, there should be several tools to analyse financial movements (interactive graphics and indicators). All visualization side should

be developed as a single-page application to permit the state of its screens and to allow a good performance and navigation. The design of the application should be responsive and adequate to its ends using an unified design. In the logic side, it should be developed a Broker API to support the contracts, orders, market data, portfolio, account summary and trading automation. It should be developed also a backtesting module to examine the developed strategies. All scripts can be saved, listed, updated and deleted. Logic side must be developed with a technology that enables the software to run with an adequate performance using an event-driven architecture. The entire solution should be developed without a server side in order to get best latency results.

## 3.5 Mind Map

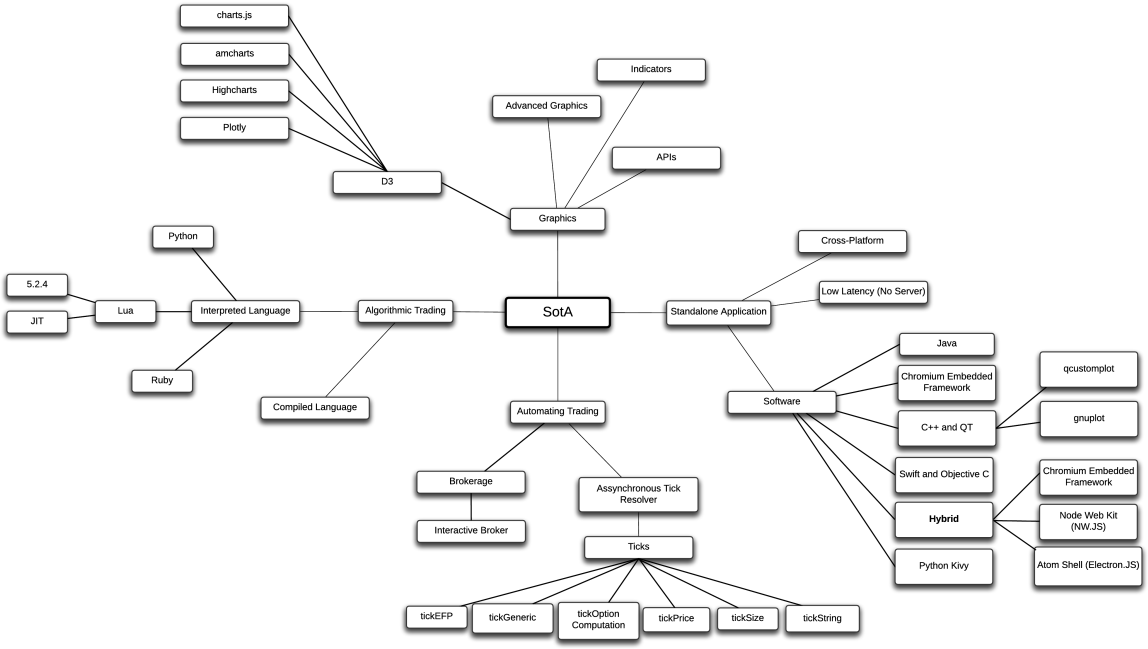


Figure 3.10: SotA Mind Map

During the State of the Art and Proposed Solution survey, it were explored many development alternatives. The principal paths were Algorithmic Trading, Standalone Application, Automatic Trading and Graphics. In Algorithmic Trading, the main focus was an interpreted language. The advantages of an interpreted language are platform independence, dynamic typing, smaller executable program size (since implementations has the flexibility to choose the instruction code) and dynamic scoping. The studied interpreted languages were Lua, Python and MQL. Afterwards, there was an effort to choose the right technology to build a standalone application. The main focus was to create a cross-platform to fight nonexistence of trading software in Linux/OSX and low-latency quality. The software/technologies/frameworks that were studied were Java, C++/QT, hybrid frameworks such as CEF and Electron.JS. It was also studied a way to automate trading with the Lua scripts and the ticks used to do it. In the end, many D3 based JavaScript graphics frameworks were analysed in order to get the

best financial stocks charts. Several financial APIs were examined too.

## 3.6 Chapter Summary

In this chapter it were presented the technologies that were studied to implement the prototype. After the definition of the functional requirements, use cases and wireframes it was possible to elaborate an architecture of the proposed solution.

# 4 Implementation

The current chapter describes the implementation of the proposed solution. The implementation contains the reasoning that was made to transform the proposed solution into a prototype. Several modules are explained in this chapter as Electron, IB API, Lua IDE, execution orders, trading analysis and others. Besides, there is a subsection where it is explained the process of iTrading installation.

## 4.1 Adopted Architecture

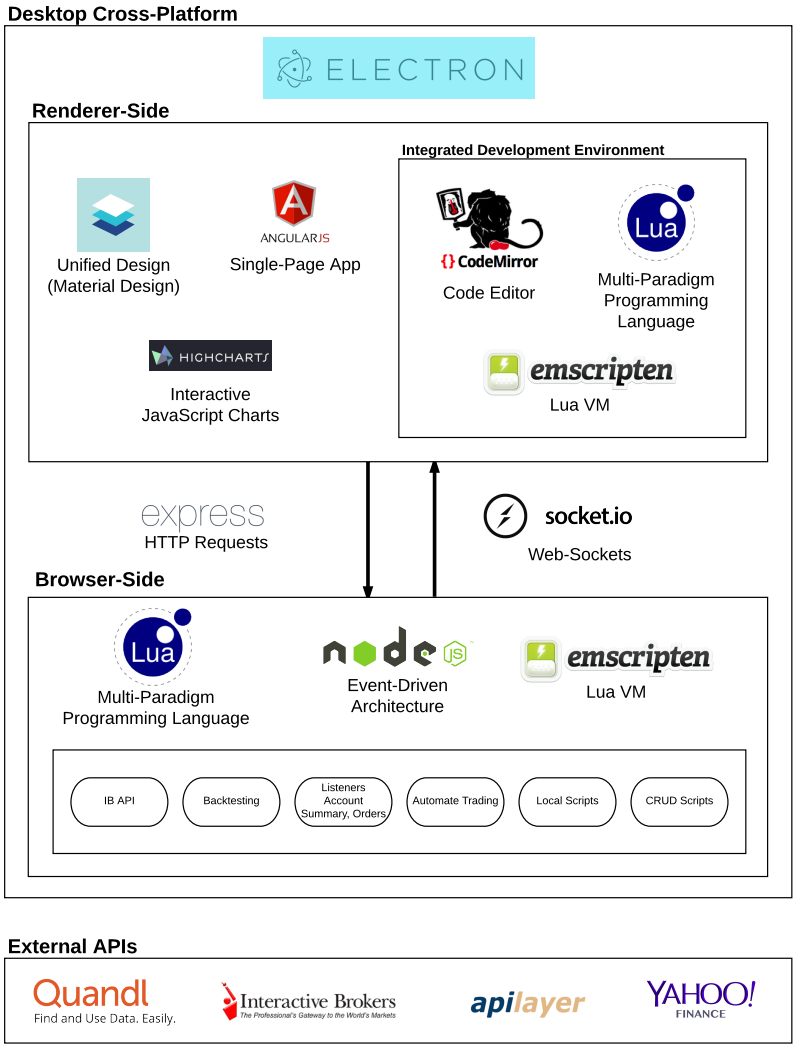


Figure 4.1: Adopted Architecture

The prototype developed for this dissertation had a base development technology based on Atom Shell, formerly known as Electron. With Electron, it was possible to develop iTrading as a cross-platform desktop application. The entire development was made with web technologies, using JavaScript, AngularJS, Google Material, an interactive JS charts (Highcharts) framework, emscripten (Lua VM) and others.

Electron is formed by Node.JS and Chromium. As a result, it is possible to have automatic updates on the view side, crash reporting, Windows/Mac-OS/Linux installers, debugging/profilling, native menus and notifications. iTrading prototype was structured in three main parts:

- **Browser-side** - responsible for business logic and data access on Interactive Broker;

- **Renderer-side** - responsible for User Interface (UI) rendering of iTrading application;
- **Modules** - the bridge between browser-side and renderer-side, it also helps to control application lifecycle.

The most relevant file in iTrading is *package.json*, which defines the entry point of the application, it is the first script that will be executed by Electron runtime. In iTrading, it is feasible to have different JavaScripts files that can be executed on browser-side or renderer-side depending on where they are called. It provides a way to use Node modules on renderer-side including Document Object Model (DOM) APIs for the page. Electron gives some extra built-in modules for developing native desktop applications. Some modules are only accessible on the browser side, some are only available on the renderer-side, and some can be used on both sides. To launch an iTrading application it's mandatory to run Electron with the path of the application (listing 4.1). When the command is executed, OS will create a new instance of Electron (browser process) that will load the script specified in the *package.json*. From this point, the script will be in charge. A renderer process will be created for each window that was specified in browser code (figure 4.2). In iTrading it was done a particular app module to control the lifecycle of the application (figure A.1, listing A.1).

```
atom.exe test-app
```

Listing 4.1: Execution process.

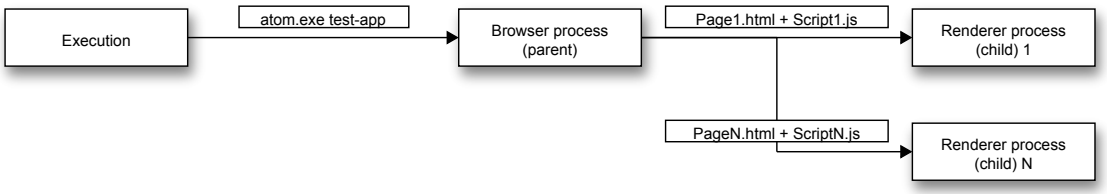


Figure 4.2: Electron Execution Process [67].

Electron offers two special modules for communication between renderer and browser sides. The IPC was used to instantiate automatic trading process on browser-side.

- **IPC** - (available on both sides) based on Chromium IPC (example A.2);
- **Remote** - (available on renderer side) it is a remote method invocation of browser's process objects (example A.3)

### 4.1.1 Event-driven Architecture

Node.JS [68] is an open-source, cross-platform runtime environment for developing, typically, server-side Web applications. The runtime environment interprets JS using Google V8 JavaScript engine. On the Node.JS (Browser-Side) it were developed some modules, such as IB API, Backtesting, Listeners to Account Summary and Orders, Automatic Trading, create/read/update/delete Scripts, among others. Node.JS has an event-driven architecture

capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in iTrading with many input/output operations or real-time actions.

iTrading operates on a single thread using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching. The design of sharing a single thread between all the requests that uses the observer pattern is intended for building highly concurrent applications, where any function performing I/O must use a callback. In order to accommodate the single-threaded event-loop, Node.JS utilizes the libuv [69] library that in turn uses a fixed-sized threadpool that is responsible for all non-blocking asynchronous I/O operations.

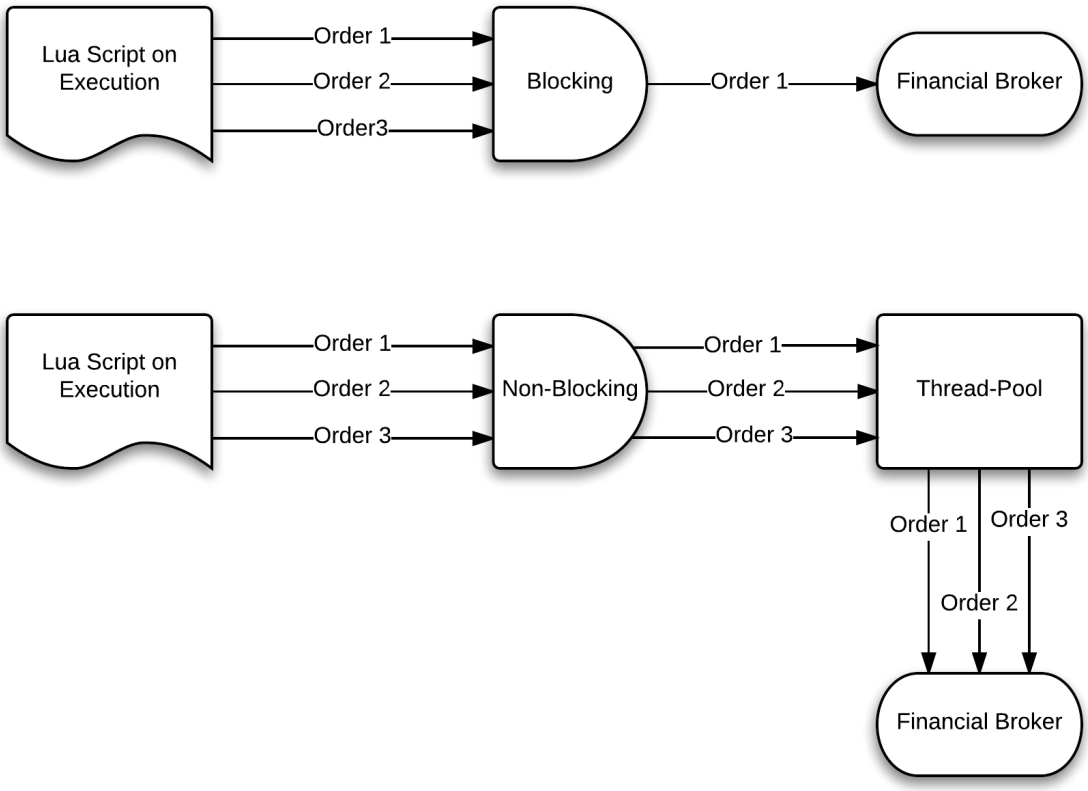


Figure 4.3: Itrading with and without Node.JS Event-Driven Architecture

### 4.1.2 Interactive Brokers API

The IB API, a fork of an open-sorce project, is located on the Browser-Side of iTrading [68]. It is a direct port of Interactive Brokers official Java client for NodeJS [46] [70] [68]. There is no C++/Java library dependency. It makes a socket connection to TWS (or IB Gateway) using net module where all messages are entirely processed in JavaScript [71]. It uses EventEmitter to pass the result back to user [72]. An example of NodeIB usage is presented in the code snippet B.1.

```
.connect()
.disconnect()
.calculateImpliedVolatility(reqId, contract, optionPrice, underPrice)
.calculateOptionPrice(reqId, contract, volatility, underPrice)
.cancelAccountSummary(reqId)
.cancelCalculateImpliedVolatility(reqId)
.cancelCalculateOptionPrice(reqId)
.cancelFundamentalData(reqId)
.cancelHistoricalData(tickerId)
.cancelMktData(tickerId)
.cancelMktDepth(tickerId)
.cancelNewsBulletins()
.cancelOrder(id)
.cancelPositions()
.cancelRealTimeBars(tickerId)
.cancelScannerSubscription(tickerId)
.exerciseOptions(tickerId, contract, exerciseAction, exerciseQuantity, account,
    override)
.placeOrder(id, contract, order)
.replaceFA(faDataType, xml)
.reqAccountSummary(reqId, group, tags)
.reqAccountUpdates(subscribe, acctCode)
.reqAllOpenOrders()
.reqAutoOpenOrders(bAutoBind)
.reqContractDetails(reqId, contract)
.reqCurrentTime()
.reqExecutions(reqId, filter)
.reqFundamentalData(reqId, contract, reportType)
.reqGlobalCancel()
.reqHistoricalData(tickerId, contract, endDateTime, durationStr, barSizeSetting,
    whatToShow, useRTH, formatDate)
.reqIds(numIds)
.reqManagedAccts()
.reqMarketDataType(marketDataType)
.reqMktData(tickerId, contract, genericTickList, snapshot)
.reqMktDepth(tickerId, contract, numRows)
.reqNewsBulletins(allMsgs)
.reqOpenOrders()
.reqPositions()
.reqRealTimeBars(tickerId, contract, barSize, whatToShow, useRTH)
.reqScannerParameters()
.reqScannerSubscription(tickerId, subscription)
.requestFA(faDataType)
.setServerLogLevel(logLevel)
```

Listing 4.2: NodeIB API

### Events

```
// General
```

```
.on('error', function (err, data))
.on('result', function (event, args)) // exclude connection
.on('all', function (event, args)) // error + connection + result

// Connection
.on('connected', function ())
.on('disconnected', function ())
.on('received', function (tokens, data))
.on('sent', function (tokens, data))
.on('server', function (version, connectionTime))

// Result
.on('accountDownloadEnd', function (accountName))
.on('accountSummary', function (reqId, account, tag, value, currency))
.on('accountSummaryEnd', function (reqId))
.on('bondContractDetails', function (reqId, contract))
.on('commissionReport', function (commissionReport))
.on('contractDetails', function (reqId, contract))
.on('contractDetailsEnd', function (reqId))
.on('currentTime', function (time))
.on('deltaNeutralValidation', function (reqId, underComp))
.on('execDetails', function (reqId, contract, exec))
.on('execDetailsEnd', function (reqId))
.on('fundamentalData', function (reqId, data))
.on('historicalData', function (reqId, date, open, high, low, close, volume,
   barCount, WAP, hasGaps))
.on('managedAccounts', function (accountsList))
.on('marketDataType', function (reqId, marketDataType))
.on('nextValidId', function (orderId))
.on('openOrder', function (orderId, contract, order, orderState))
.on('openOrderEnd', function ())
.on('orderStatus', function (id, status, filled, remaining, avgFillPrice, permId,
   parentId, lastFillPrice, clientId, whyHeld))
.on('position', function (account, contract, pos, avgCost))
.on('positionEnd', function ())
.on('realtimeBar', function (reqId, time, open, high, low, close, volume, wap,
   count))
.on('receiveFA', function (faDataType, xml))
.on('scannerData', function (tickerId, rank, contract, distance, benchmark,
   projection, legsStr))
.on('scannerDataEnd', function (tickerId))
.on('scannerParameters', function (xml))
.on('tickEFP', function (tickerId, tickType, basisPoints, formattedBasisPoints,
   impliedFuturesPrice, holdDays, futureExpiry, dividendImpact, dividendsToExpiry))
.on('tickGeneric', function (tickerId, tickType, value))
.on('tickOptionComputation', function (tickerId, tickType, impliedVol, delta,
   optPrice, pvDividend, gamma, vega, theta, undPrice))
.on('tickPrice', function (tickerId, tickType, price, canAutoExecute))
.on('tickSize', function (tickerId, sizeTickType, size))
.on('tickSnapshotEnd', function (reqId))
.on('tickString', function (tickerId, tickType, value))
.on('updateAccountTime', function (timeStamp))
.on('updateAccountValue', function (key, value, currency, accountName))
```

```
.on('updateMktDepth', function (id, position, operation, side, price, size))
.on('updateMktDepthL2', function (id, position, marketMaker, operation, side,
    price, size))
.on('updateNewsBulletin', function (newsMsgId, newsMsgType, newsMessage,
    originatingExch))
.on('updatePortfolio', function (contract, position, marketPrice, marketValue,
    averageCost, unrealizedPNL, realizedPNL, accountName))
```

Listing 4.3: NodeIB Events

#### Builders

```
// Contract
.contract.combo(symbol, currency, exchange)
.contract.forex(symbol, currency)
.contract.future(symbol, expiry, currency, exchange)
.contract.option(symbol, expiry, strike, right, exchange, currency)
.contract.stock(symbol, exchange, currency)

// Order
.order.limit(action, quantity, price)
.order.market(action, quantity)
.order.stop(action, quantity, price)
.order.stopLimit(action, quantity, limitPrice, stopPrice)
```

Listing 4.4: NodeIB Builders

### 4.1.3 Automatic Trading

With this design all I/O requests to Interactive Broker, API Layer, Yahoo Finance will be non-blocking allowing a much better performance (figure 4.3). The figure 4.4 shows the architecture implemented in iTrading for automatic trading. Every script that is associated to a specific market data, e.g. *stock, FB* for all change that occur on Facebook stock regarding *EFP*, *Generic*, *Option Computation*, *Price*, *Size* or *String* will trigger an event that goes to Node.JS Event Loop. For all events that exist in the queue, the script associated to the tick will be executed.

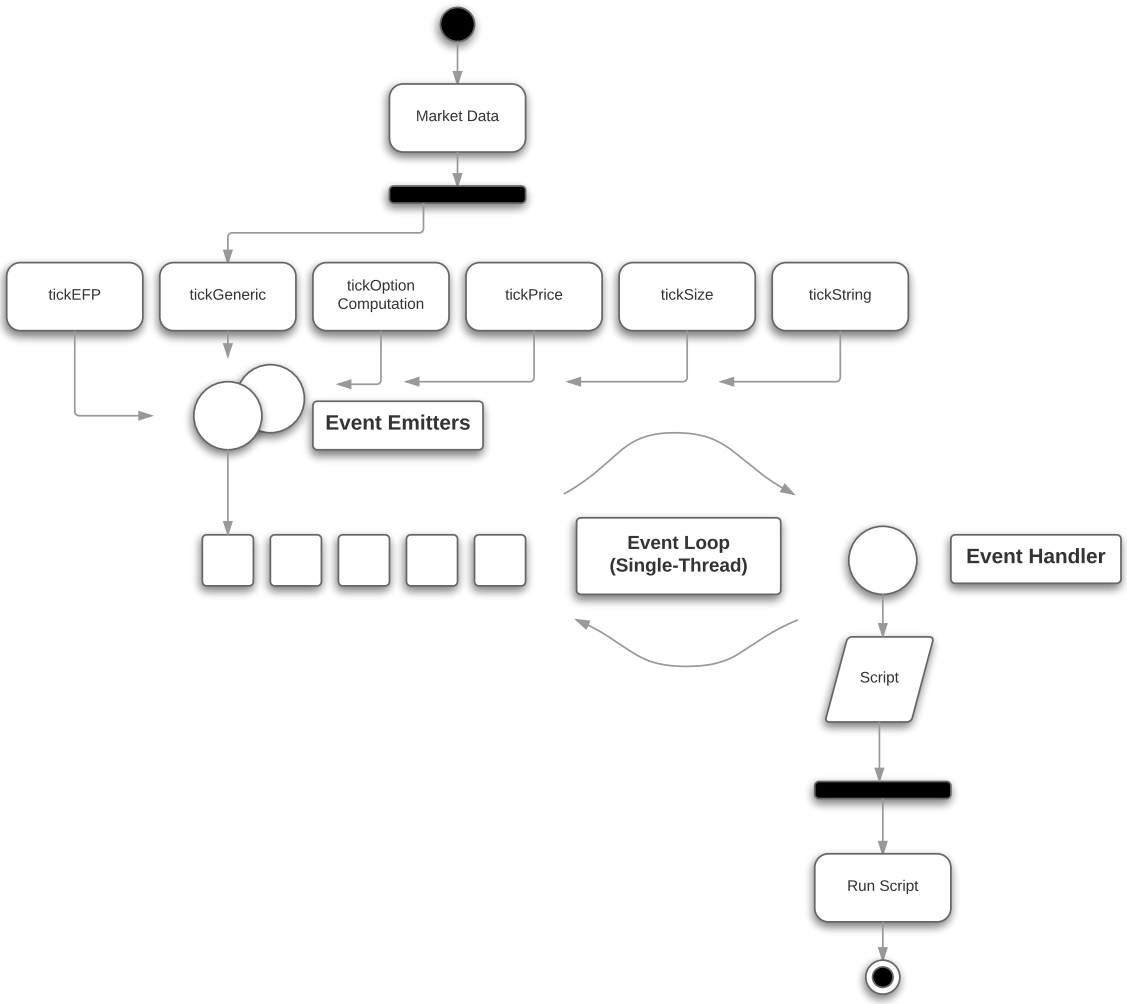


Figure 4.4: Automatic Trading

### 4.1.4 Multi-Paradigm Programming Language

Lua was chosen as the Algorithmic Trading language of iTrading. Lua Virtual Machine (VM) is an experiment in running Lua on the web, by porting the Lua C implementation to JavaScript using Emscripten (more information about Emscripten see appendice B) [73] [74] [75]. All the Lua 5.2.4 codebase written in portable C is compiled to JavaScript, including a full incremental GC and everything else. It fits in 170K of gzipped JavaScript. It is possible to run C compiled to JavaScript at a speed approaching that of a native build, which means that is possible to run C code that happens to implement a VM at high speed as well [76].

The developed algorithmic trading IB API was done to support six types of financial contracts and four types of orders (table 4.1). Everytime a script is executed, iTrading will run Lua VM. If the compiler got one or more methods, which send an order to IB, those methods will be executed on IB API browser-side, figure 4.5). There are several possible API' methods (tables: A.1, A.2, A.3, A.4, A.5, A.6). As described in figure 4.6, after perform

*Execute Script*, it will be executed on Lua VM. If the script developed has a Lua error, it will return to Lua IDE with the error message, otherwise it will run on Lua VM. If there are any IB API' methods, it will be triggered an event for each one to IB API via HTTP request. In IB API these events will be sent to Interactive Brokers Server via TCP using a callback. The response received from IB Server to the callback will be given to IB API, which will send it to Lua VM. In turn, it will send to Lua IDE as a successful or not emitted order (diagram figure 4.6). A more detailed functionality of IB API is described in section 4.1.2.

```
1
2  SYMBOL = 'AAPL'
3  EXCHANGE = 'SMART'
4  CURRENCY = 'USD'
5  ACTION = 'SELL'
6  QUANTITY = 1
7  PRICE = 0.01
8  LIMITPRICE = 0.5
9  STOPPRICE = 1
10
11 if condition then
12   stock_limit(SYMBOL, EXCHANGE, CURRENCY, ACTION, QUANTITY, PRICE)
13 end
14
```

Figure 4.5: iTrading IDE - Example Stock Limit Order

| Contracts | Orders |
|---|---|
| Stocks | Market |
| Forex | Limit |
| CFDs | Stop |
| Combo | Stop Limit |
| Futures | - |
| Options | - |

Table 4.1: Contracts and Orders available in IB API

| Stock Contract |
|---|
| stock_limit (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, PRICE) |
| stock_market (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY) |
| stock_stop (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, PRICE) |
| stock_stoplimit (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, LIMITPRICE, STOPPRICE) |

Table 4.2: Stock Contract - IB API's Methods

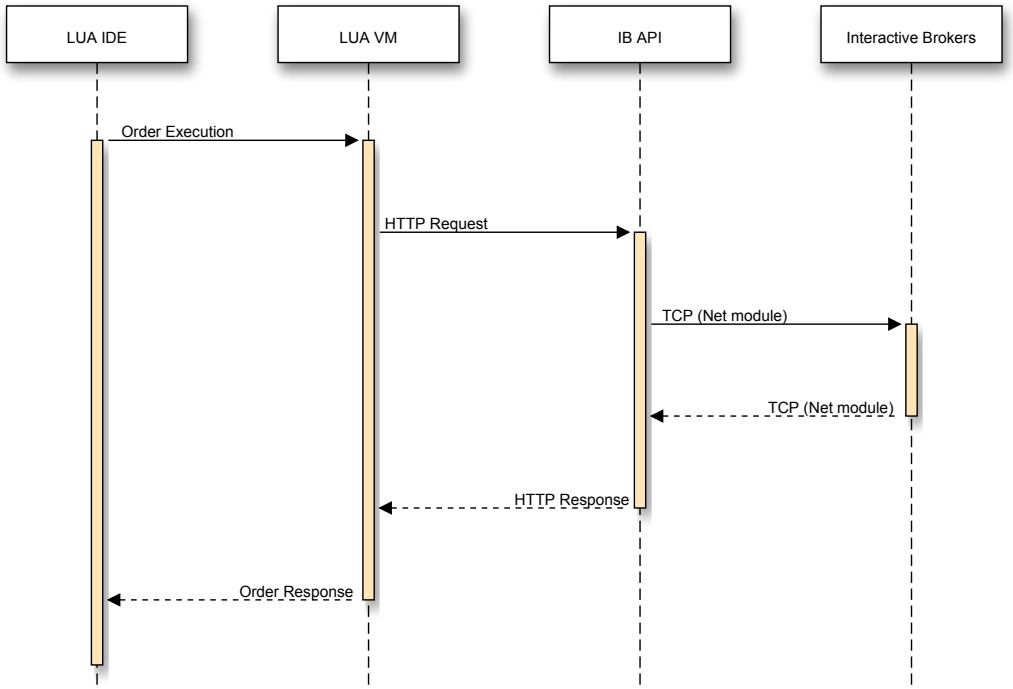


Figure 4.6: Financial Orders Sequence

### 4.1.5 Backtesting Module

It was developed a backtesting module to ensure that the investor understood the strategy effectively. In order to do that, it was developed a module which receives three parameters: the script to backtest, start date and end date of backtesting. The external API's used were Currency Layer API for FOREX and Yahoo Finance API for the rest. Those APIs return the close quotes of the date interval specified. For each day, it is analysed all financial products that are explicit in the script. Before starting backtesting, its amount is zero. After the execution (it can take 10 seconds) the total of all profits and losses of all contracts/orders will be added, giving the *backtesting amount*. In the end, the trader will understand how successful his strategy was. If the resulting backtesting amount is above zero the amount will be in green, otherwise it will appear in red (figure 4.7). The illustration 4.8 shows the sequence of backtesting.

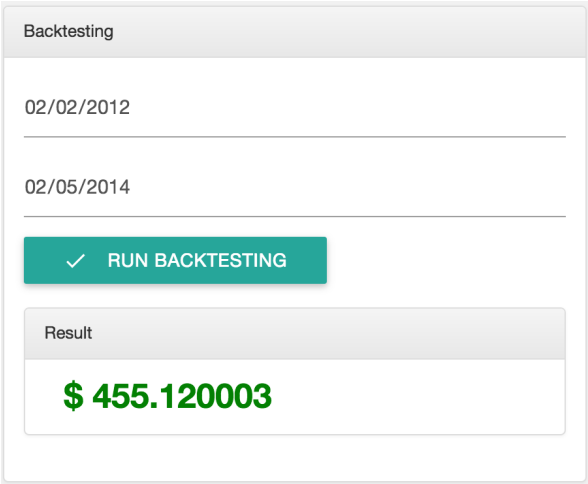


Figure 4.7: Backtesting Screenshot

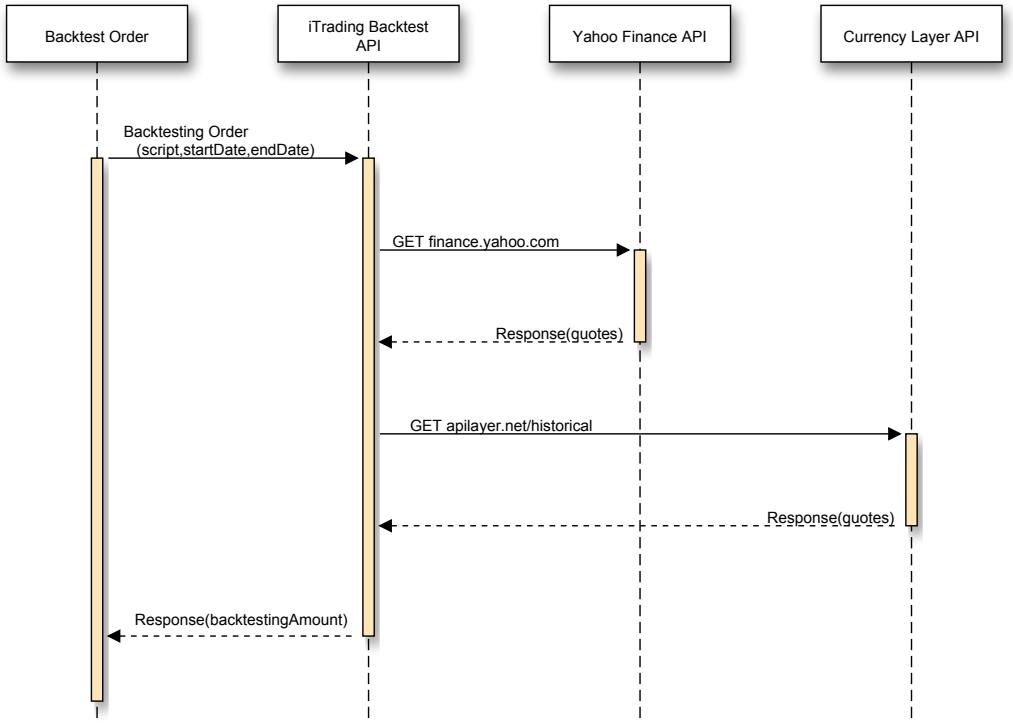


Figure 4.8: Backtesting Sequence Diagram

### 4.1.6 Asynchronous I/O API

To develop an interactive communication session between the browser-side and the renderer-side it was used WebSockets. Using this API, it was possible to send messages to browser-side and receive event-driven responses without having to poll the browser-side for a reply.

The interfaces to do such are:

- **WebSocket:** the first interface for connecting to a WebSocket server and then sending and receiving data on the connection.
- **The event:** sent by the WebSocket object when the connection closes.
- **MessageEvent:** the event sent by the WebSocket object when a message is got from the server.

Socket.IO [77] was chosen to implement WebSockets (example in listing 4.5 and 4.6). It is an event-based bi-directional communication layer for realtime web applications. It abstracts multiple transports, including Asynchronous Javascript and XML (AJAX) long-polling and WebSockets, into a single API (figure 4.9).

```
var server = require("net").createServer();
var io = require("socket.io")(server);

var handleClient = function (socket) {
    // we've got a client connection
    socket.emit("account_summary", {user: "itrading", text: "account_summary
        emit!"});
};

io.on("connection", handleClient);

server.listen(8080);
```

Listing 4.5: Socket.IO on Browser-Side

```
<script src="/socket.io/socket.io.JS"></script>
<script>
    var socket = io.connect("http://localhost");
     socket.on("connect", function () {
        console.log("Connected!");
    });
</script>
```

Listing 4.6: Socket.IO on Renderer-Side

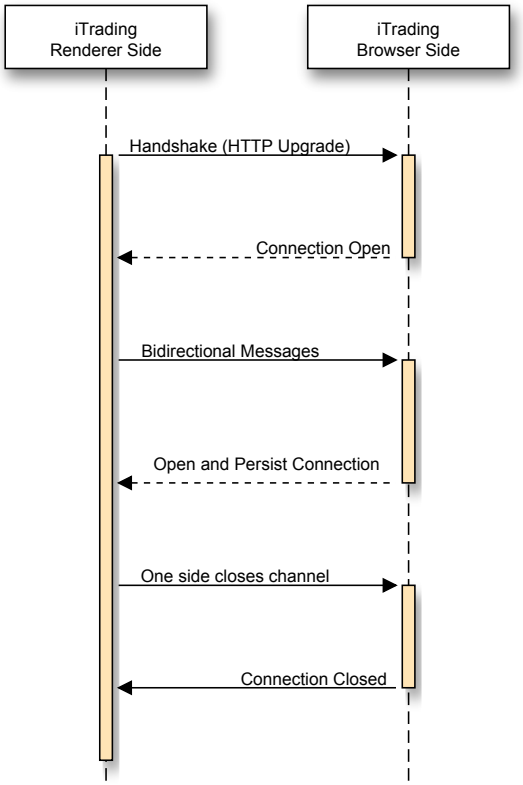


Figure 4.9: iTrading with WebSockets

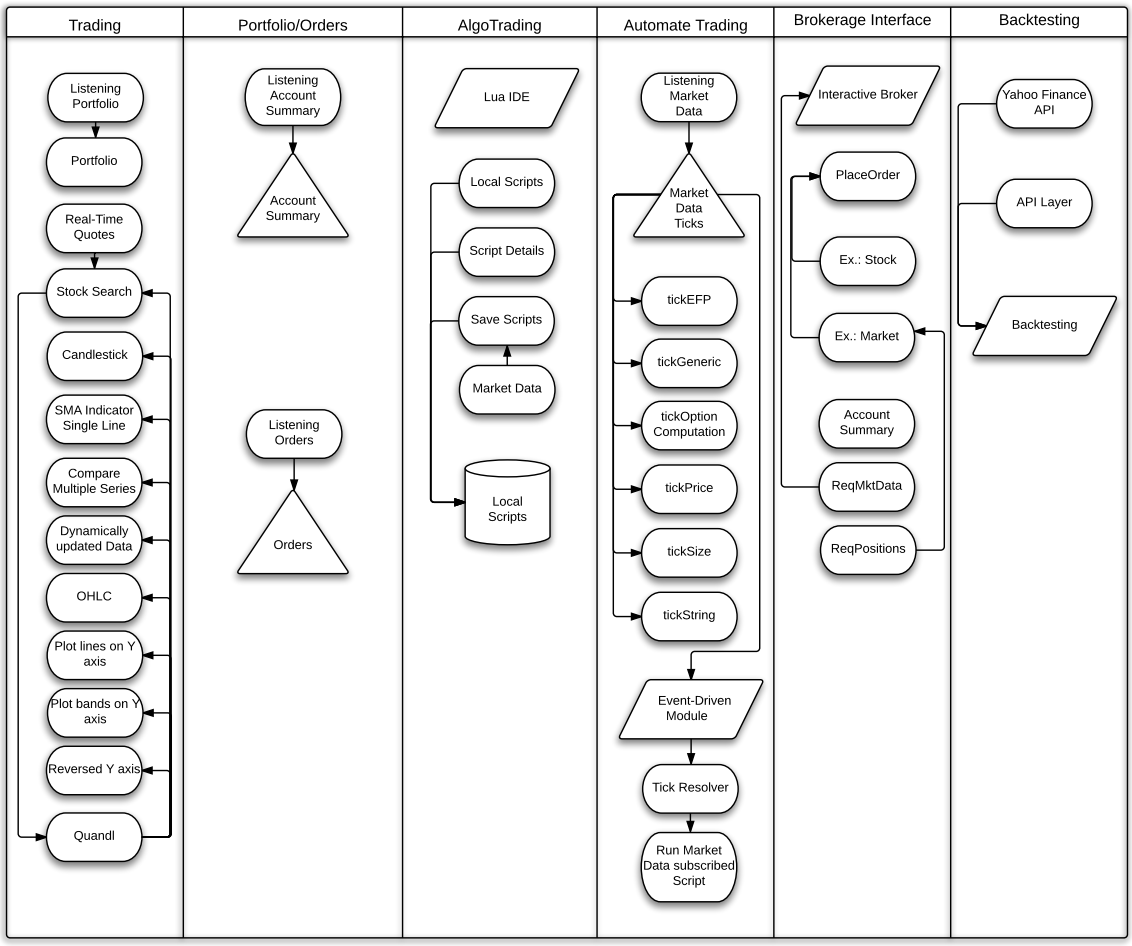


Figure 4.10: Architecture by Screens

The *trading screen* of figure 4.10 is divided by portfolio, real-time quotes, stock search, financial graphics and their source API Quandl. The portfolio is a summary of the account details of the trader. Some information like *Available Funds*, *Buying Power*, *Accrued Cash* and others are available on this area. The goal of this area is to get informed in the analysis screen about the users' gains or losses. Besides, in portfolio it is possible to analyse fluctuation of financial markets with several graphics like candlestick, SMA Indicator, different products comparation, dynamically updated data, OHLC, plot lines on Y axis, plot lines on X axis or reversed Y axis. All these graphics are fed by Quandl API (figure 4.11).

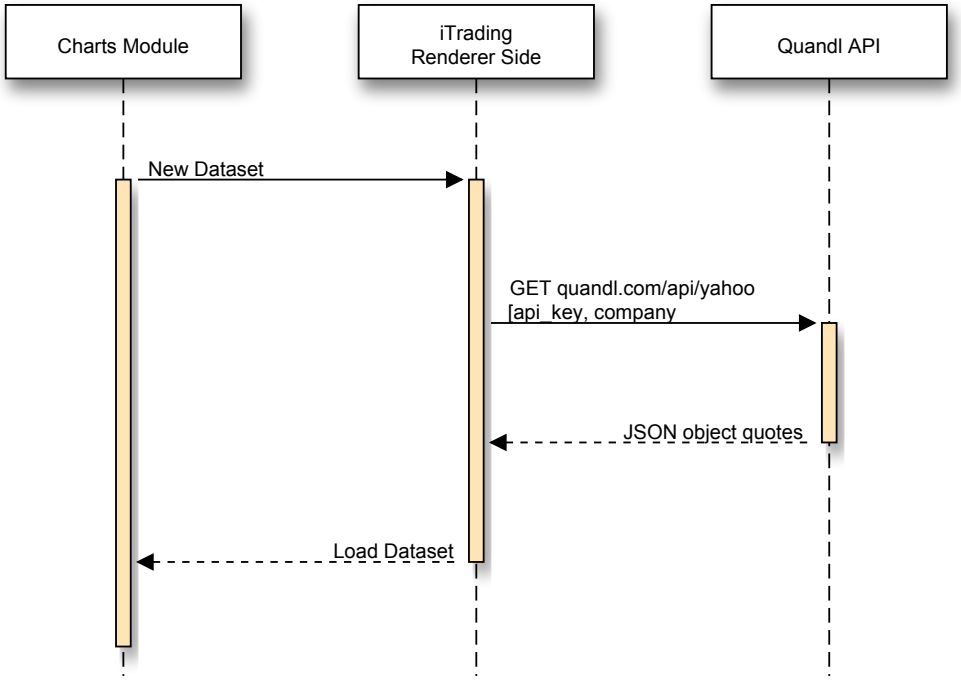


Figure 4.11: Charts - Quandl API Sequence Diagram

Quandl is a search engine for numerical data [78]. It offers access to several million financial, economic and social datasets. Quandl indexed data from multiple sources allows users to find and download it in various formats. All Quandl's data are accessible via an API [79]. A data request could be done anonymously (listing 4.7) or authenticated (listing 4.8). To get unlimited data requests it was used the authenticated mode.

```
https://www.quandl.com/api/v3/datasets/WIKI/FB.csv
```

Listing 4.7: Quandl API anonymous data request

```
https://www.quandl.com/api/v3/datasets/WIKI/FB.csv?api_key=HHU&\%FGDK__93
```

Listing 4.8: Quandl API authenticated data request

The *Account Summary* screen of figure 4.10 is where the account summary of the user is, a more detailed portfolio of the losses and gains. In *Orders* there is the registry of the past orders of IB API.

### 4.1.7 Integrated Development Environment

In *AlgoTrading* screen there is the integrated development environment. To create it was used CodeMirror. CodeMirror is a versatile text editor implemented in JavaScript for the browser [80]. It is specialized for editing code and comes with a number of language modes

and addons, which implement more advanced editing functionality. A rich programming API and a CSS theming system were used to fit iTrading functionality.

Angular.JS was used to get iTrading renderer-side as a single-page application. Angular.JS aims to simplify the development by providing model-view-controller (MVC) and model-view-view-mode (MVVM) architectures for client-side, along with components commonly used in rich Internet applications [81]. In iTrading, Angular.JS was important to create the flow between tabs without loading every screen everytime a user clicks in one. iTrading saves every state that exists between screens. For example, if a user is writing a script and wants to check a specific stock graphic he can do it without losing what he has been doing. Besides, the HTTP requests were made with it too (figure 4.12). Angular.JS works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular.JS interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The goals in Angular.JS were:

- to decouple DOM manipulation of an application from application logic. The difficulty of this is dramatically affected by the way the code is structured.
- to decouple the client side of an application from the server side. This allows development work to progress in parallel and allows the reuse of both sides.
- to provide structure for the journey of building an application: from designing the UI to writing the business logic to testing.

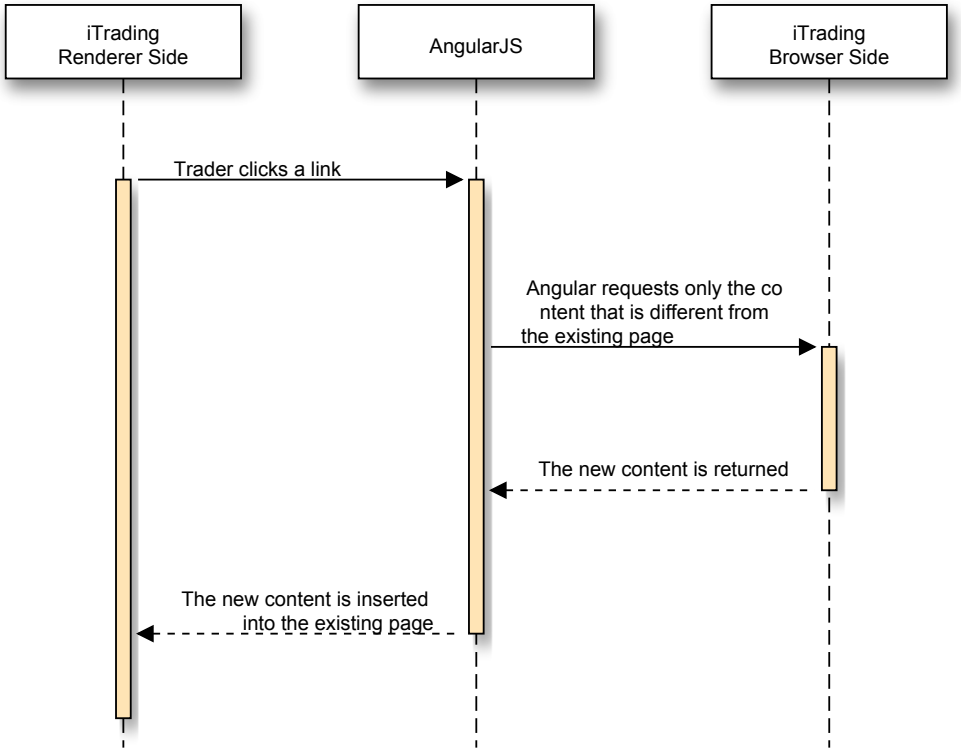


Figure 4.12: Angular Single Application Sequence Diagram

### 4.1.8 Unified Design

In order to get a unified design and user experience there was used Material Design, build by Google [82]. This design couples classic beliefs of successful design along with novelty and technology. The metaphor of material represents the connection between space and motion. The plan is that the technology is inspired by paper and ink and is utilized to promote creativity and change. Surfaces and edges give familiar visual cues that enable users to quickly understand the technology beyond the physical world.

Elements and components such as grids, typography, color, and imagery are not simply visually pleasing, but also produce a sense of hierarchy, meaning, and focus. Emphasis on diverse actions and components create a visible guide for users. By giving both feedback and familiarity, this enables the user to fully engage him or herself into unfamiliar technology. The motion includes consistency and flows, in addition to giving users extra unconscious information regarding objects and transformation. It provides a default styling that incorporates custom components. Moreover, they refined animations and transitions to provide a smoother experience for developers.

## 4.2 iTrading Download Page

In order to promote iTrading application it was developed a single-page website of iTrading (figure 4.13). The main purpose of this page was to explain the functionalities of iTrading and to catch the attention of financial enthusiastic to use it and have feedback, bugs, improvements, among others. It was made a track of the webpage with Google Analytics in order to get statistical data about accesses/users/session duration (figure 4.14), downloads (figure A.3), source flow, among others. This page was spread on several social networks such as Reddit (35% of the traffic), Facebook (5%), Hacker News (3%), Caldeirão da Bolsa (2%) and others. There were more than 910 views to this page

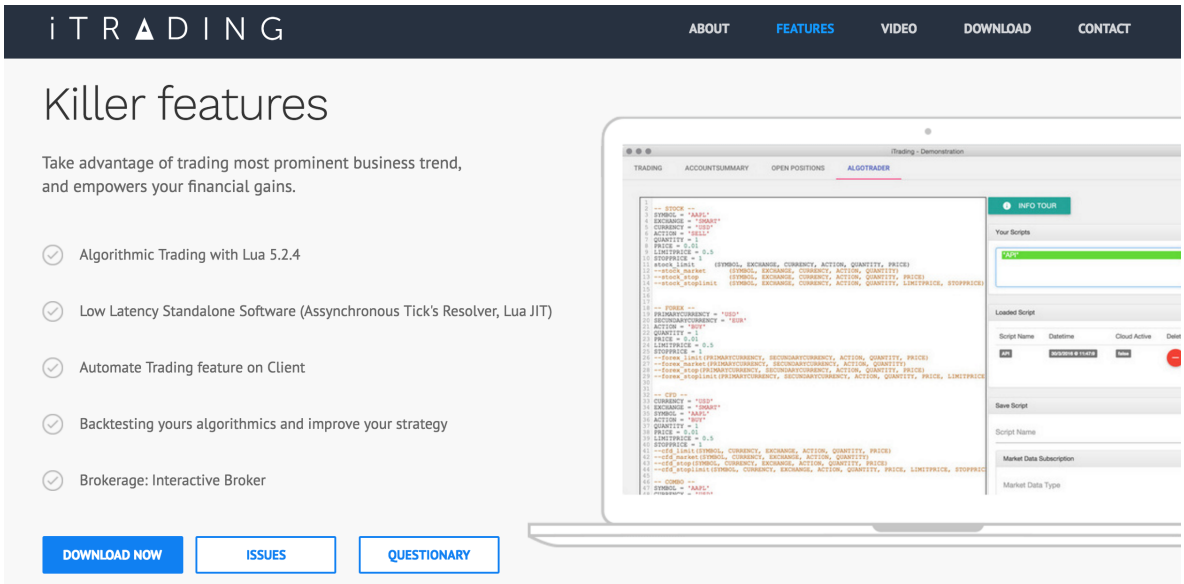


Figure 4.13: iTrading Download Page

There were made 71 downloads until 31 of March. The majority of downloads were made for Windows-platforms with 43%, 32% and 25% for OSX and Linux platforms, respectively (figure A.3).

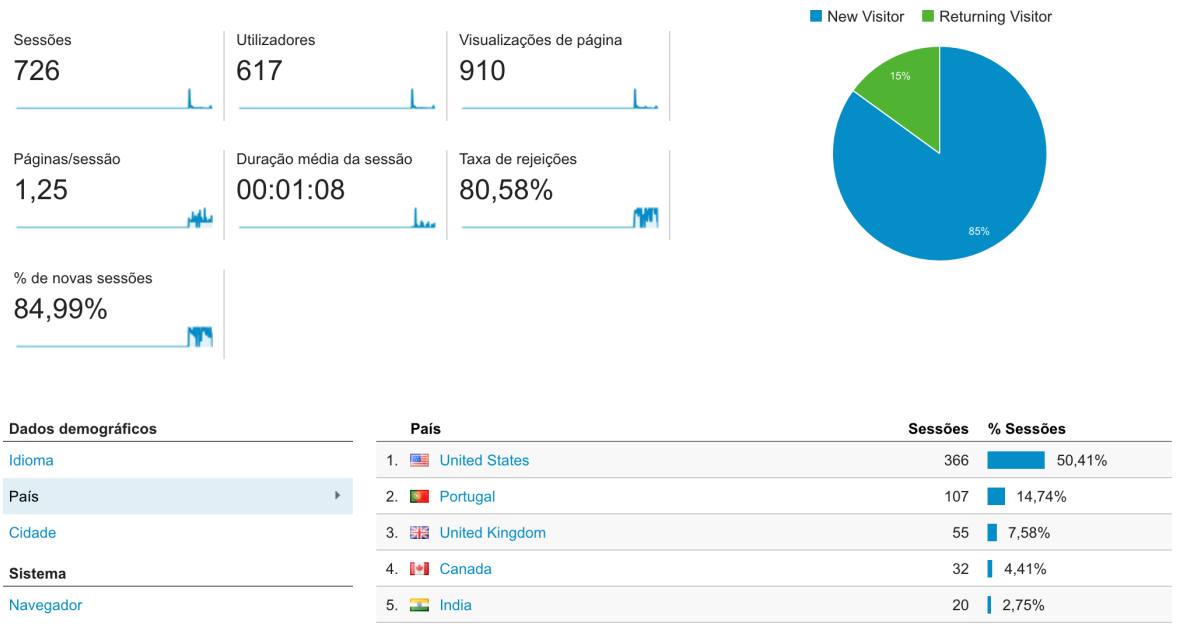


Figure 4.14: Google Analytics - General Data

## 4.3 Installation Guide

After the conclusion of iTrading development, it was created an installation guide (figure 4.15). This guide provides instructions on how to install and configure iTrading and Interactive Brokers. The content of this manual is divided into five sections:

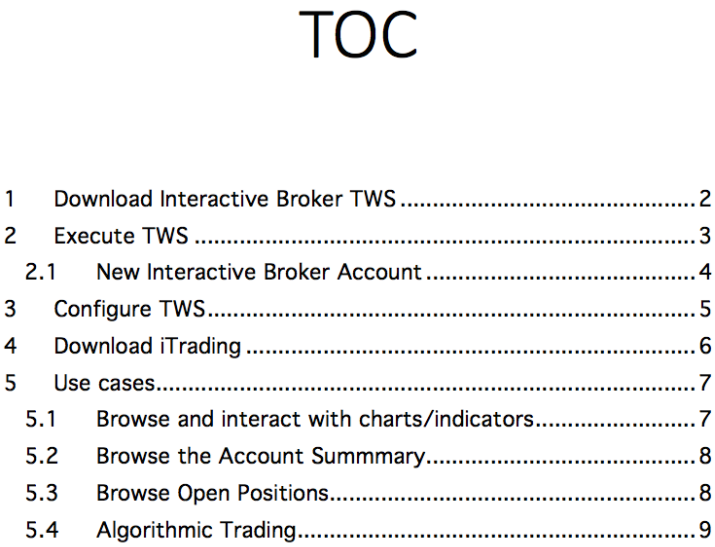


Figure 4.15: iTrading Installation Guide

## 4.4 Chapter Summary

In this chapter, it was presented the architecture that was developed for the prototype as well as the technologies that were used. Every single module developed was described. Besides, there are sections regarding the iTrading download page and its installation guide.

# 5 Evaluation and Results

Chapter 5 details the evaluation process (questionnaire and results), visual results with iTrading screenshots and profiling.

## 5.1 Evaluation Questionary

After developing the prototype, especially the GUI development of iTrading, it was necessary to do some usability tests, in order to evaluate the user experience. The majority of these usability tests were made by students of Aveiro's University, who were submitted to a functionality evaluation.

On the last phase of the development, there were made some experiences with the people that came to Students@DETI, an University of Aveiro's event, aiming to test iTrading programming environment and functionalities inherent.

To evaluate iTrading is was used Nielsen's Usability Heuristics for User Interface Design [83]. They are widely used by usability professionals for quickly identifying design problems in an application's human interface. Because of its simplicity and low cost, it is a preferred evaluation technique at the earliest design stages. The goal of the heuristic evaluation was to find the usability problems in design. The following ten principles were analysed by the testers using a questionnaire (figure 5.1):

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use

- Aesthetic and minimalist design
- Help users recognize, diagnose, and recover from errors
- Provision of Help and documentation

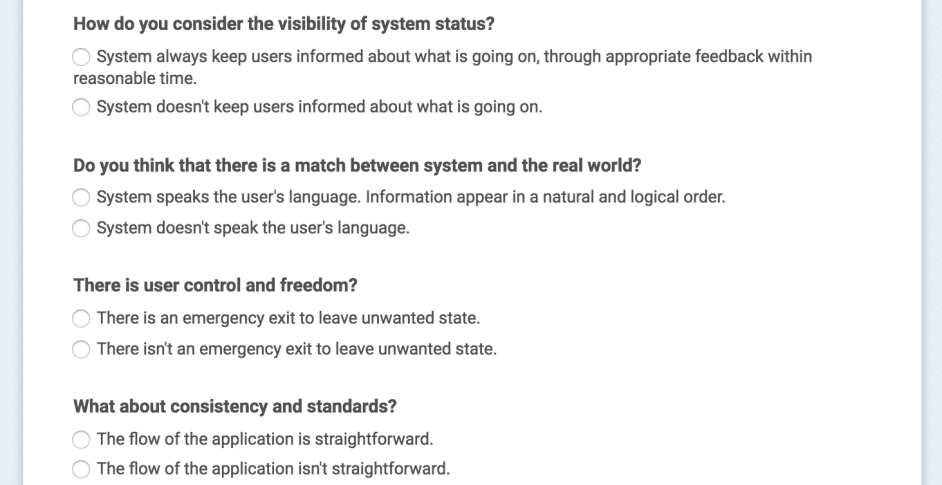


Figure 5.1: GUI Formulary

The GUI formulary was answered by 23 different persons. It is expected that this value would be significative. The majority of them were students but there is also a chief financial officer. The average age of participants was 24 years old. Only one person had experience in other trading software (Plus500 software). Next, it will be presented several graphics created based on the answers of testers.

**Average age of participants**

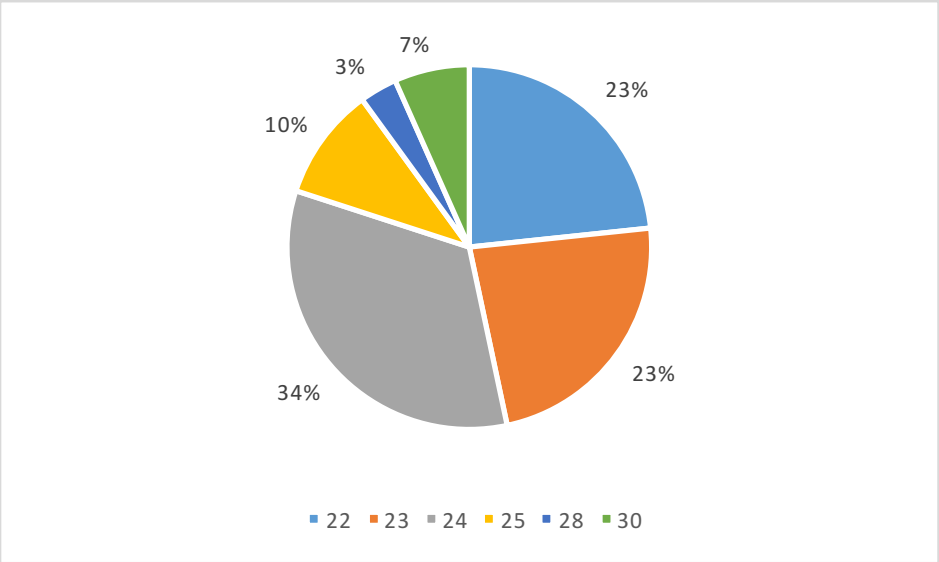


Figure 5.2: Average age of participants

The minimum age was 22 years old and the maximum was 30 years old. The average age is young and this may have an impact on the evaluating results since the majority of questionnaire group doesn't have experience in trading systems or financial markets.

### 5.1.1 Results

**Visibility of system status**

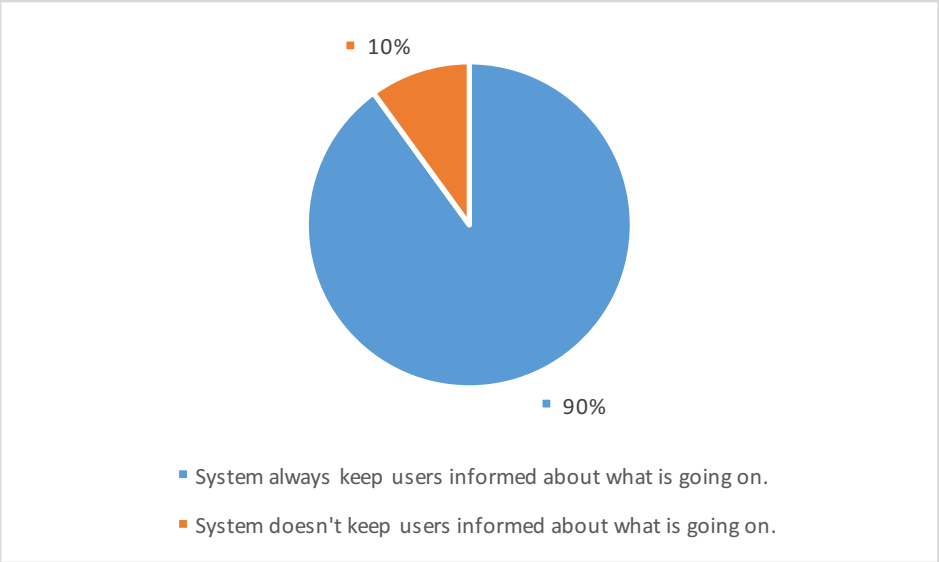


Figure 5.3: Users Feedback - Visibility of system status

According to the Nielsen's Usability Heuristics the system should always keep users informed about what is going on, through appropriate feedback within reasonable time. A larger part of the users answered positively to this heuristic. The negative answers might be because of the time that is necessary to open the application or the time to run a script with IB orders.

**Match between system and the real world**

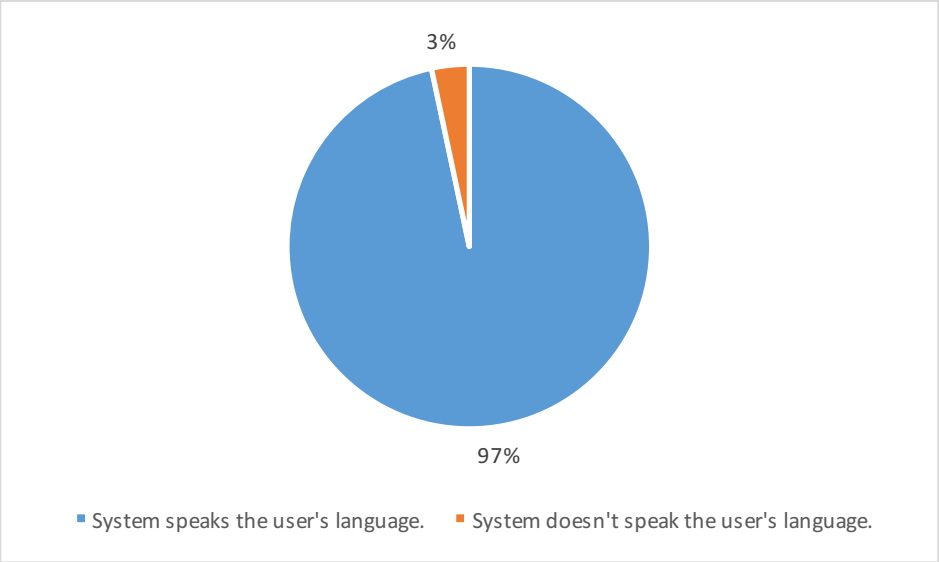


Figure 5.4: Users Feedback - Match between system and the real world

This heuristic states that the system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. It should follow real-world conventions, making information appear in a natural and logical order. The answers to these heuristics were very positive. This result should be related to simplicity of the vocabulary and terms used in application.

### User control and freedom

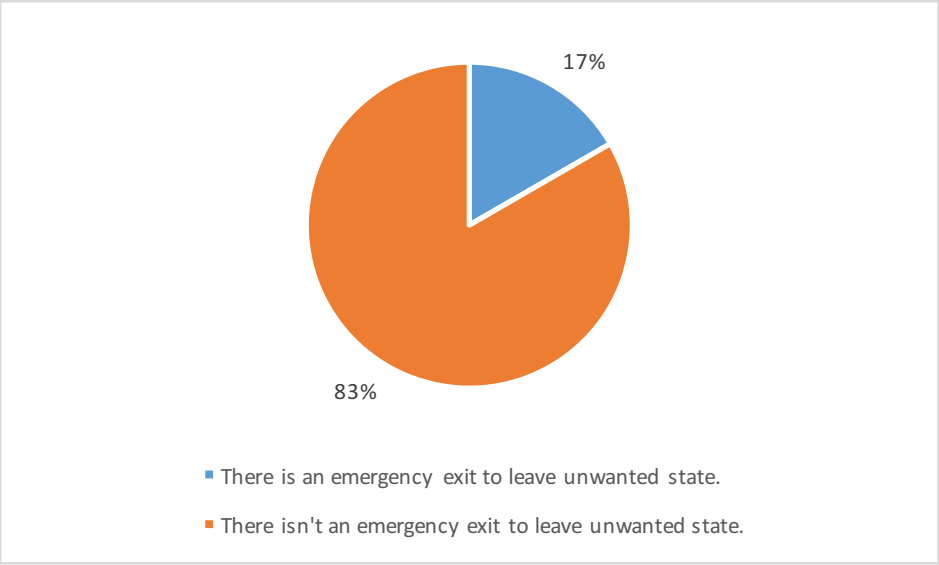


Figure 5.5: Users Feedback - User control and freedom

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. It must support undo and redo operations. The 17% of the negative answers may result of the orders that were sent to the financial broker. However, this order cannot be cancelled since it is a definitive order.

### Consistency and standards

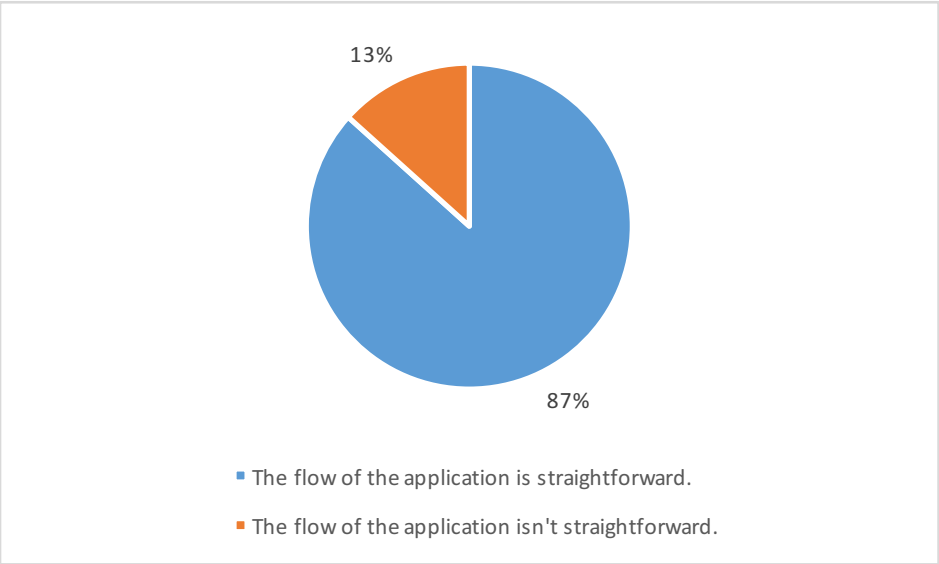


Figure 5.6: Users Feedback - Consistency and standards

This heuristic means that users should not have to wonder whether different words, situations or actions mean the same thing. In iTrading all actions and words mean different behaviours. The 13% of the users that said that the flow isn't straightforward may not understand some financial actions or vocabulary such as the difference between financial products, some terms like backtesting, limit order, stoplimit order, and others. However all the actions are documented with an interactive guide.

**Error prevention**

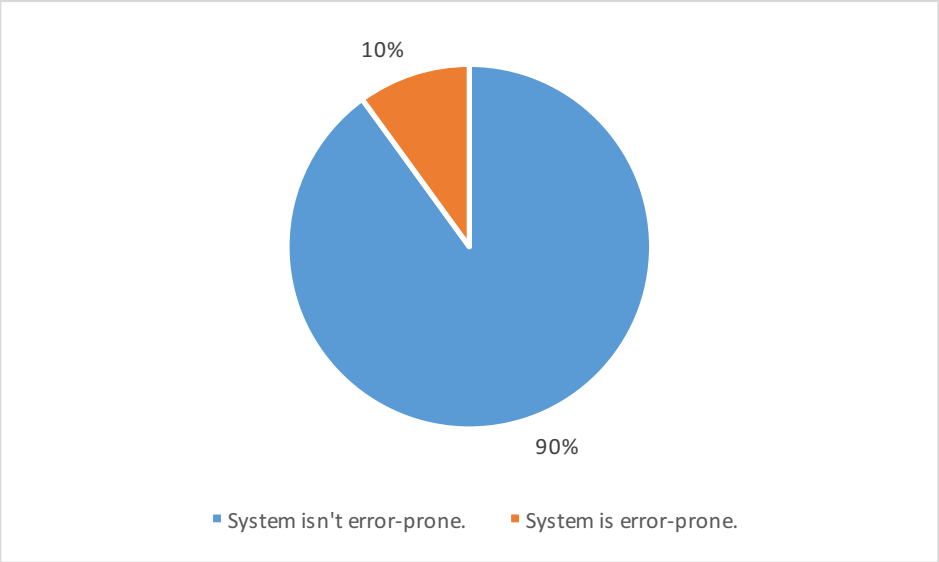


Figure 5.7: Users Feedback - Error prevention

This heuristic expresses that even better than good error messages is a careful design which prevents a problem from occurring in the first place. An application should either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action. Some people (10%) answered that the system is error-prone. It might be because of the log that IB gives after the execution of a script with a IB order.

### Recognition rather than recall

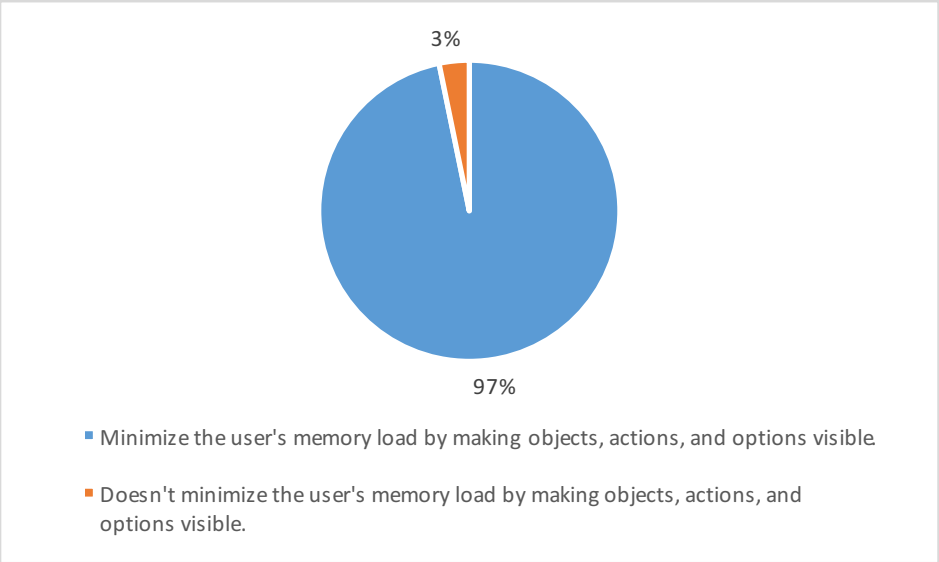


Figure 5.8: Users Feedback - Recognition rather than recall

Recognition rather than recall is meant to minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for using the system should be visible or easily retrievable whenever appropriate. Most of the people answered that the system minimizes the users' memory. This is because iTrading is a single-page application that preserves the state of all screens and doesn't load all the application in every action.

### Flexibility and efficiency of use

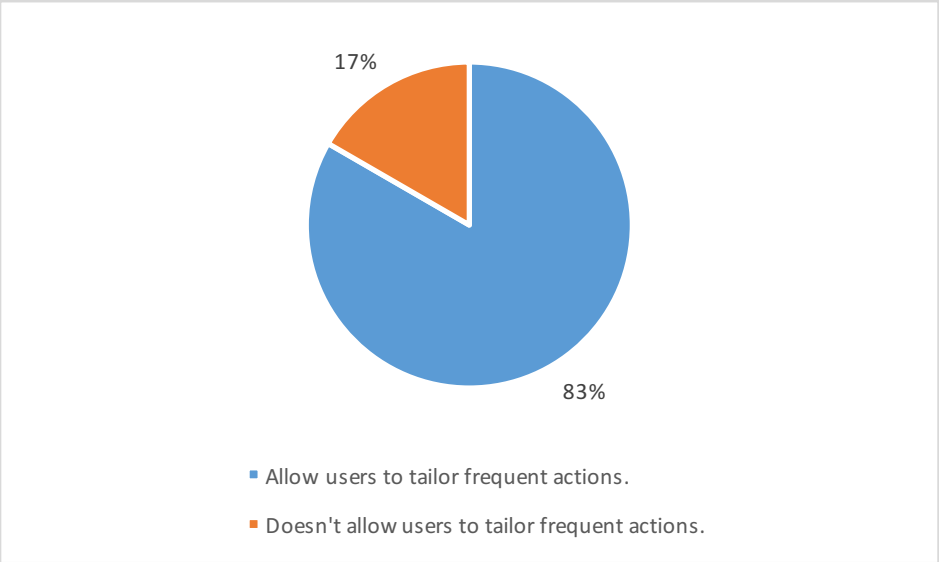


Figure 5.9: Users Feedback - Flexibility and efficiency of use

This heuristic affirms that accelerators may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Most of the users say that the application runs fluidly without the need of any shortcuts or accelerators while others claim that shortcuts are needed for expert users to execute script orders rapidly.

### Aesthetic and minimalist design

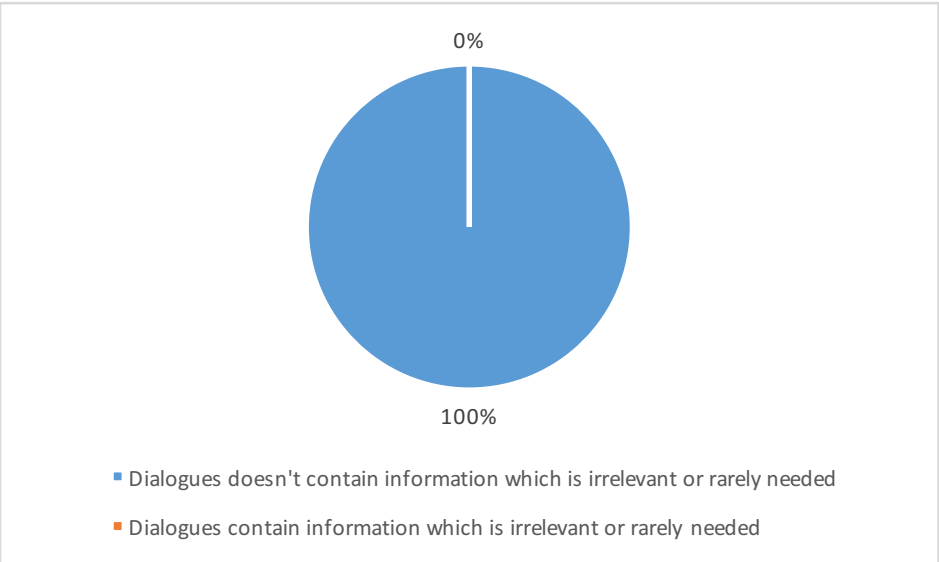


Figure 5.10: Users Feedback - Aesthetic and minimalist design

100% of the users though the dialogues contained relevant information. Every extra unit of information in a dialogue competes with the relevant units of information. The design approach was that iTrading needs to be simple and intuitive.

**Help users recognize, diagnose, and recover from errors**

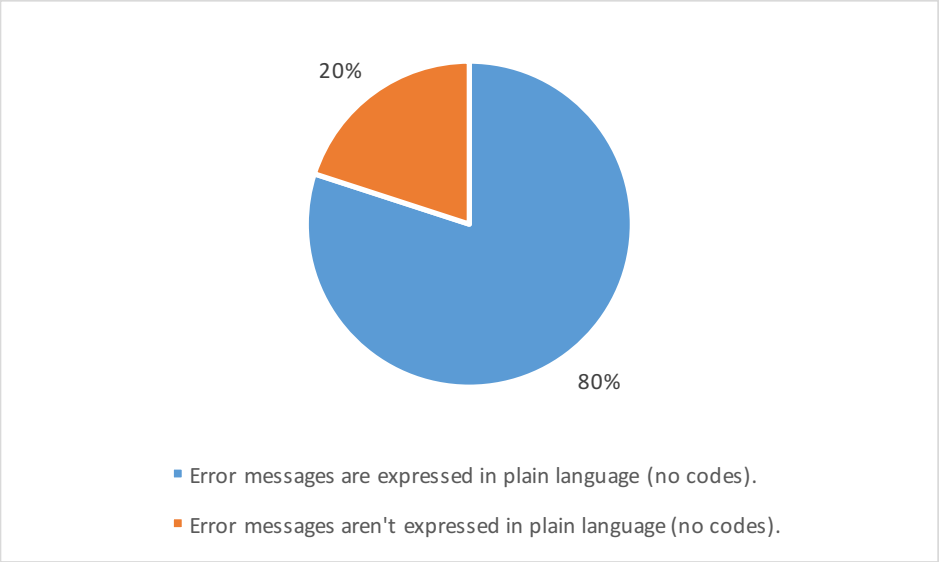


Figure 5.11: Users Feedback - Help users recognize, diagnose, and recover from errors

Help users recognize, diagnose and recover from errors means that error messages should be expressed in plain language (no codes), precisely indicating the problem, and constructively suggesting a solution. The system has an area that corresponds to IB Order result. This result is very difficult to understand and should be treated more properly.

### Provision of Help and documentation

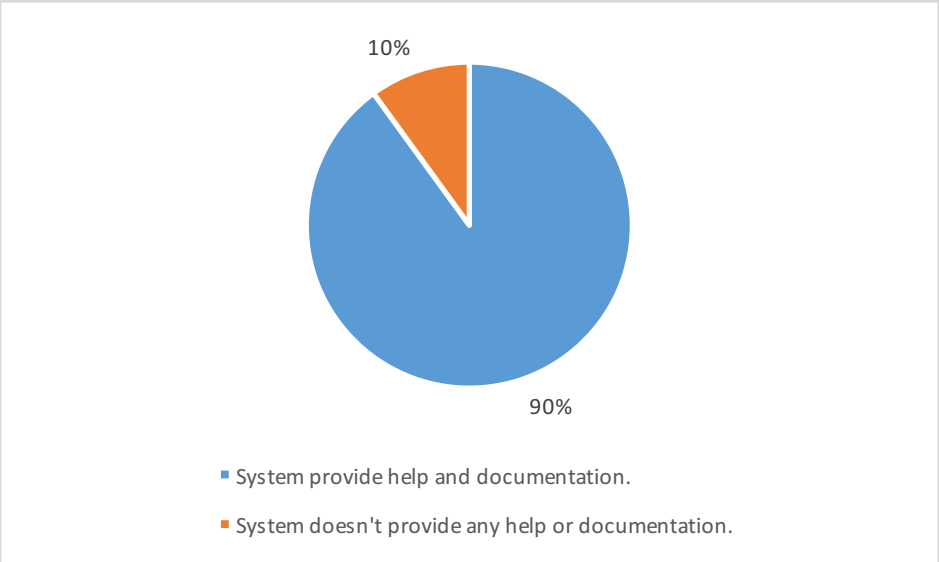


Figure 5.12: Users Feedback - Provision of help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large. The majority of testers answered that the system contains enough information related to the IB API while others found difficult to reach the documentation button.

### Latency after some action

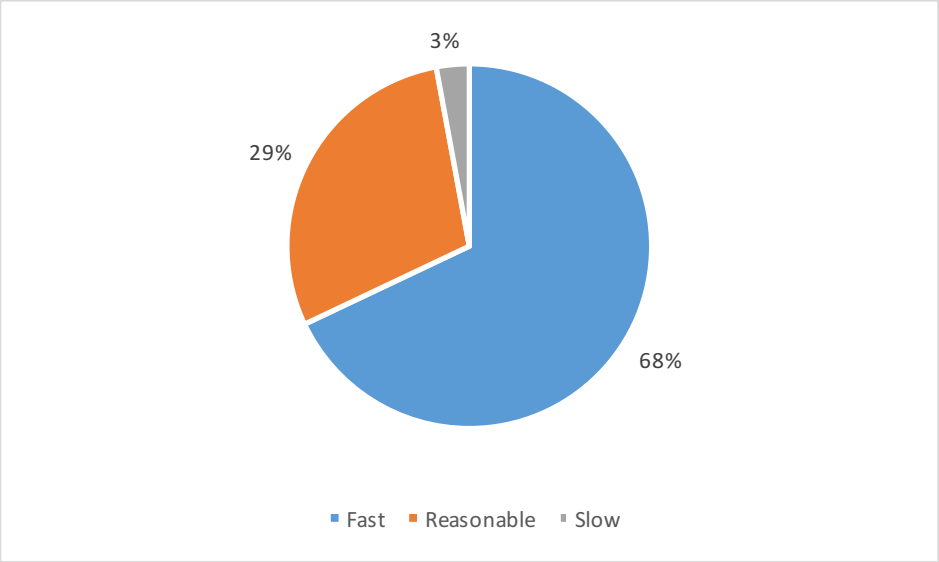


Figure 5.13: Users Feedback - Latency after some Action

Most of people said that the system is fast (68%), others (29%) said that is reasonable and very few answered that is slow (3%). The results are satisfactory. The reasons of it are that all the system was build under asynchronous technologies such as Asynchronous Javascript and XML (AJAX) and an Event-Driven architecture (Node.JS).

**Time to List Graphics**

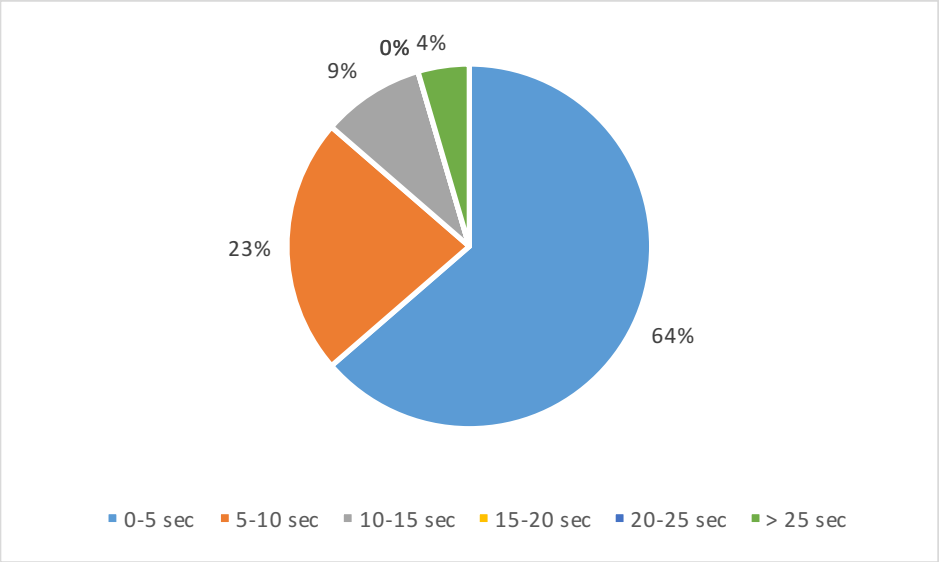


Figure 5.14: Users Feedback - Time to List Graphics

The average of the users said that 0-5 seconds were enough to list all the graphics in the trading screen. The result is good however it doesn't require much experience to accomplish this action. The test consisted into list all the graphics and choose one of them.

### Time to List Account Summary

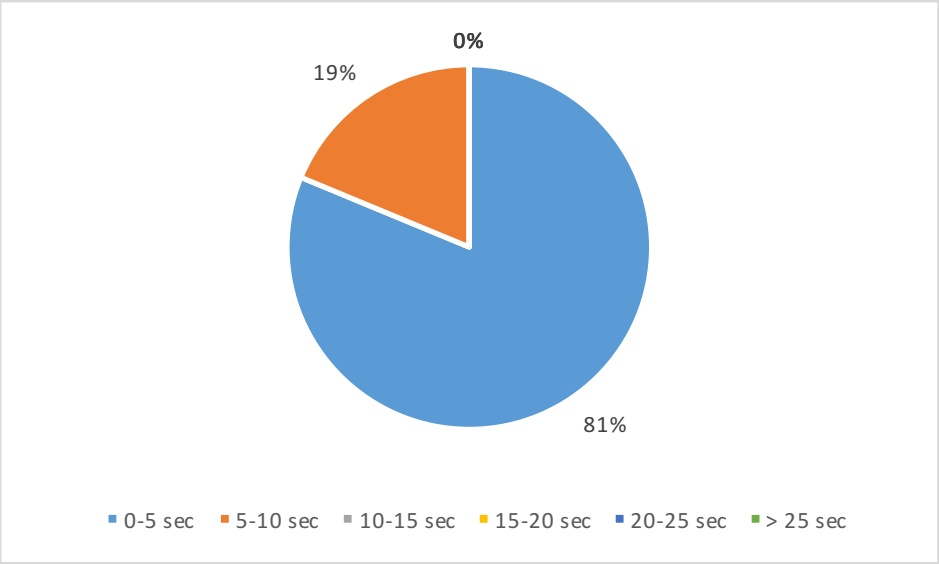


Figure 5.15: Users Feedback - Time to List Account Summary

Most of the user said that they could list account summary within 0 to 5 seconds. Others, 19% said they need 5 to 10 seconds to complete the task. The results are unexpetected since it is a easy task.

### Time to List Orders

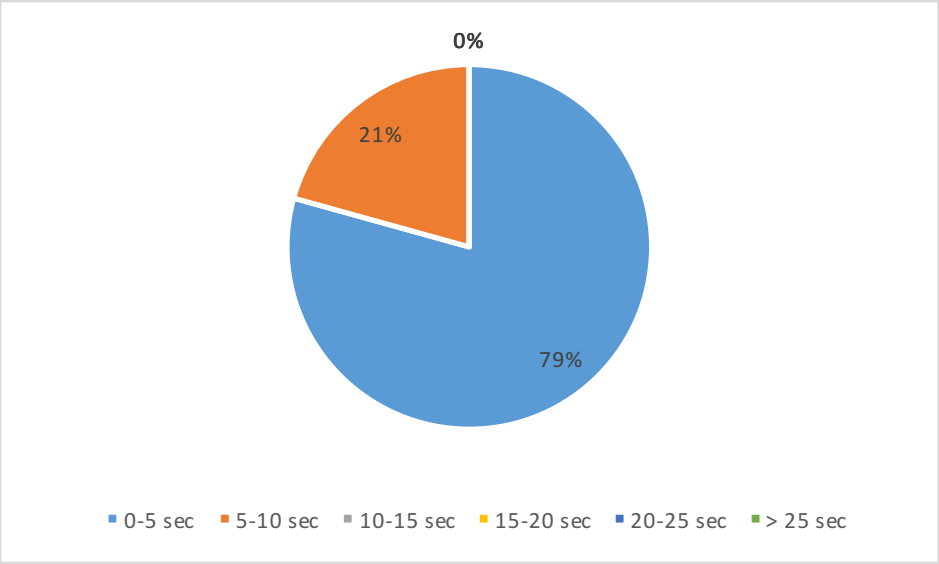


Figure 5.16: Users Feedback - Time to List Orders

79% of the users answer that 0 to 5 seconds were enough to list the orders that they've done whereas 21% of the users said they need 5 to 10 seconds. It may be due the lack of experience.

**Time to Execute a Script**

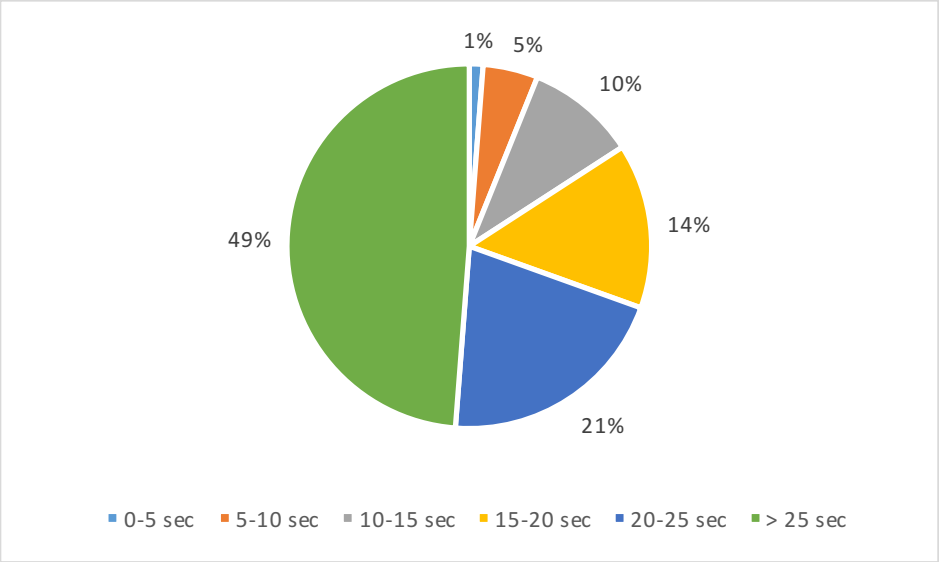


Figure 5.17: Users Feedback - Time to Execute a Script

There were very different answers regarding the time to execute a script. The average users said that 10 to 15 seconds were enough. This time could probably be minimized to 5 to 10 seconds as the user gets more experienced with the software.

**Time to Save a Script and Activate it to run Automatically**

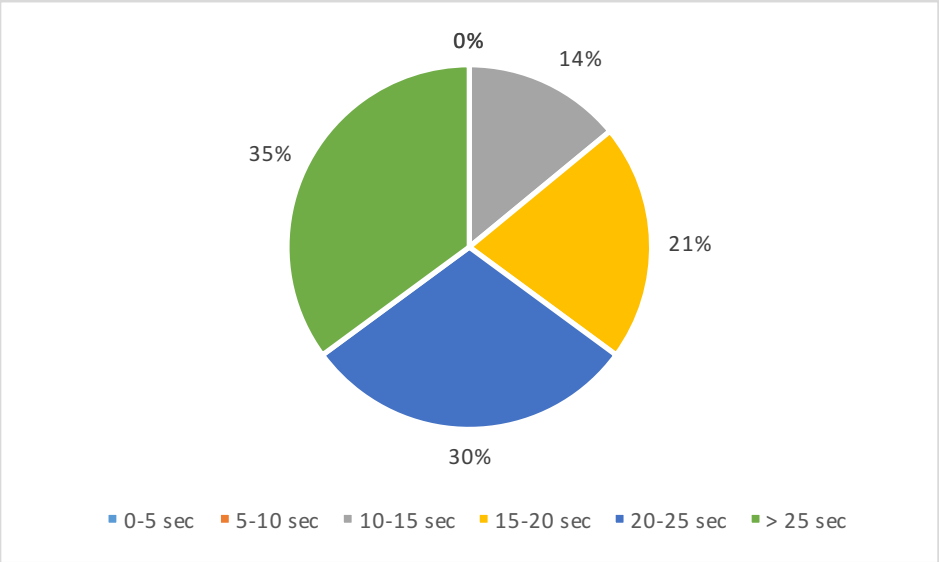


Figure 5.18: Users Feedback - Time to Save a Script and Activate it to run Automatically

Users had very different results regarding the time to save a script and activate it to run automatically. Surely, the time can be improved as they get more experience using this software. However it is an expected result.

## 5.2 Visual Result

In this section, it will be presented the visual output of iTrading application. The visual result is divided into four windows namely trading, account summary, open positions and algorithmic trading. For each one, to understand it better, it will be presented the use cases linked to it

### 5.2.1 Trading Window

Trading window is the first screen of iTrading. Its focus is analyse the market data with many interactive graphics: Candlestick; SMA Indicator Single Line Series; Compare Multiple Series; Dynamically Updated Data; OHLC; Plot Lines on Y/X Axis and Reversed Y Axis. It is very easy to zoom in or zoom out the data. Besides, it is possible to search financial products and get portfolio summary.

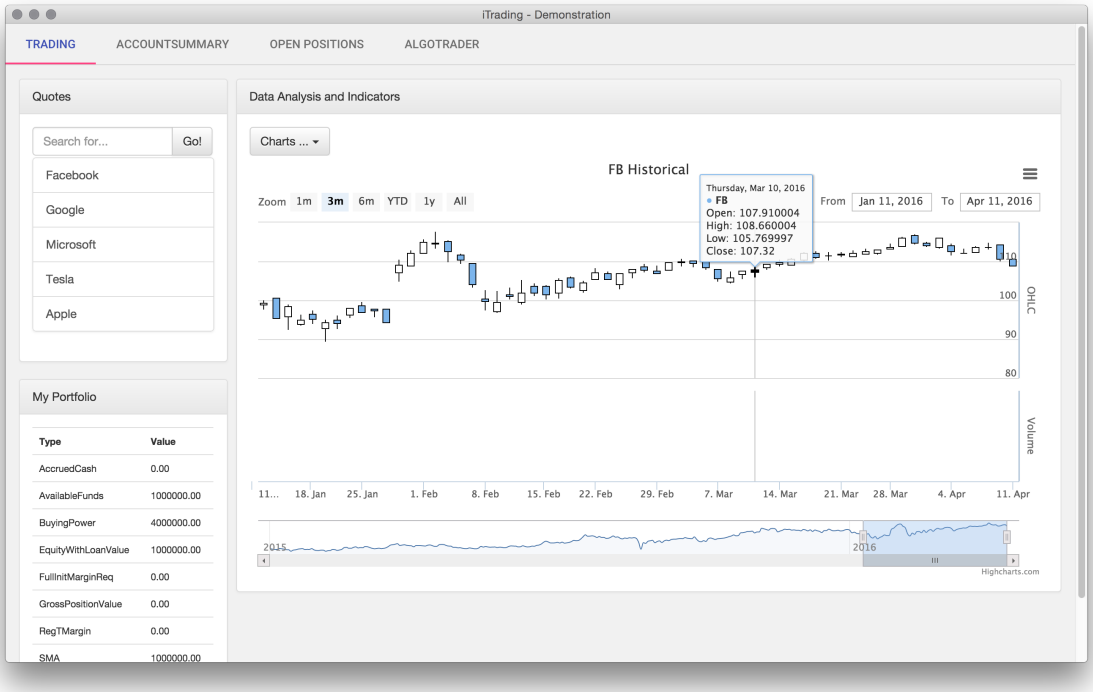


Figure 5.19: iTrading - Trading Window - Candlestick Graphic

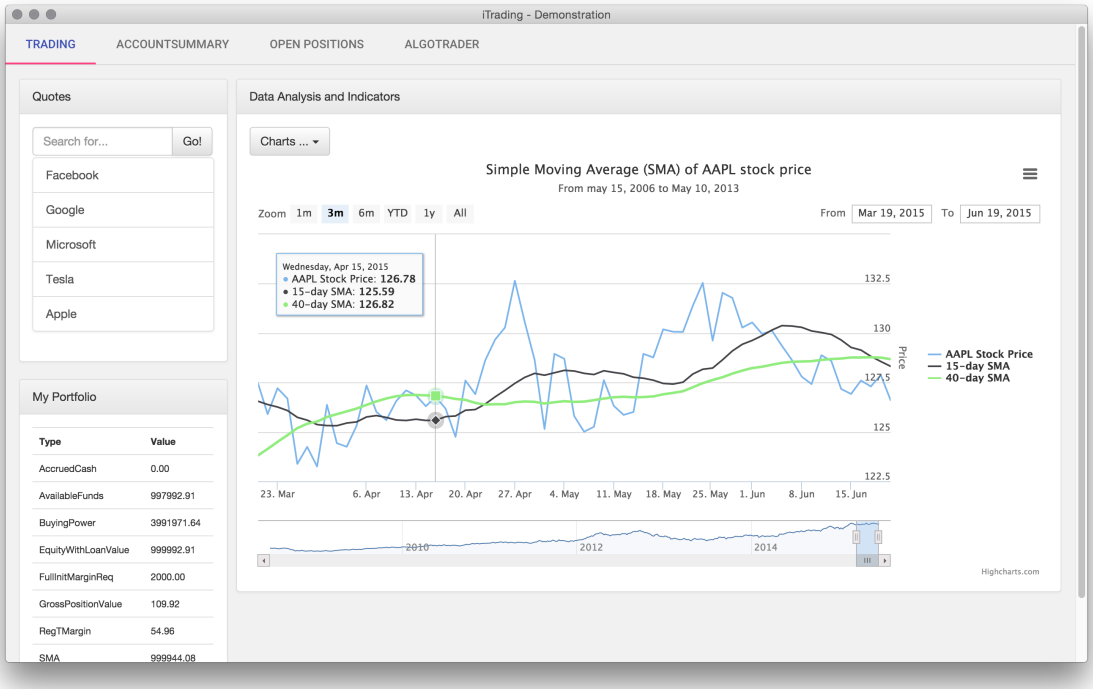


Figure 5.20: iTrading - Trading Window - SMA Indicator Graphic

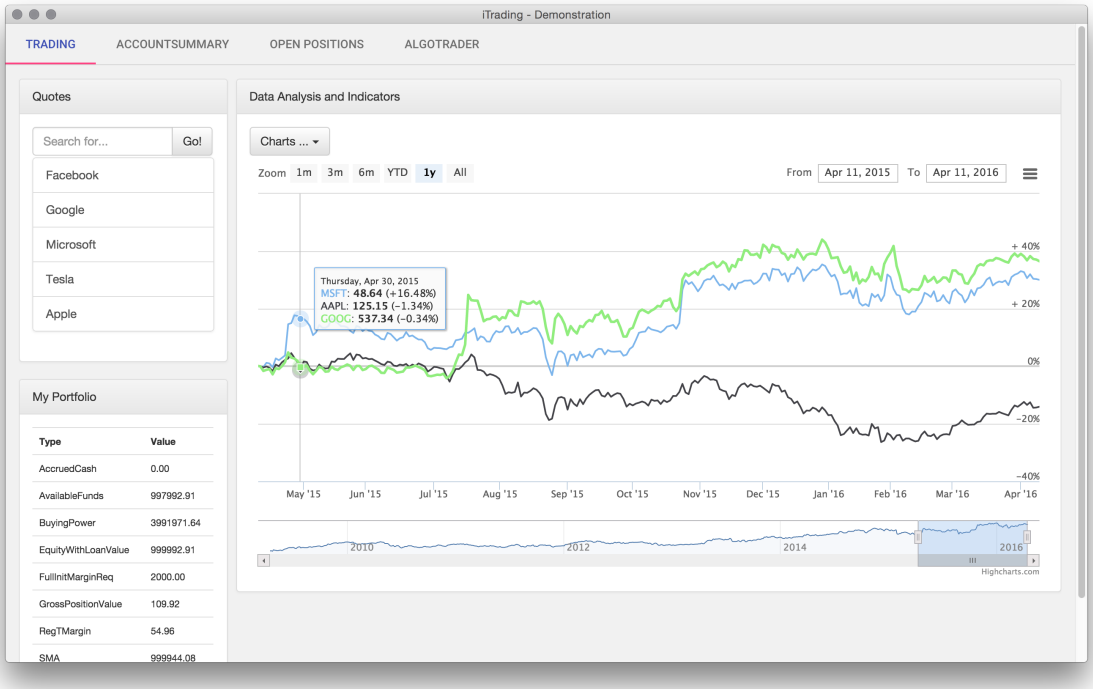


Figure 5.21: iTrading - Trading Window - Multi-Graphics

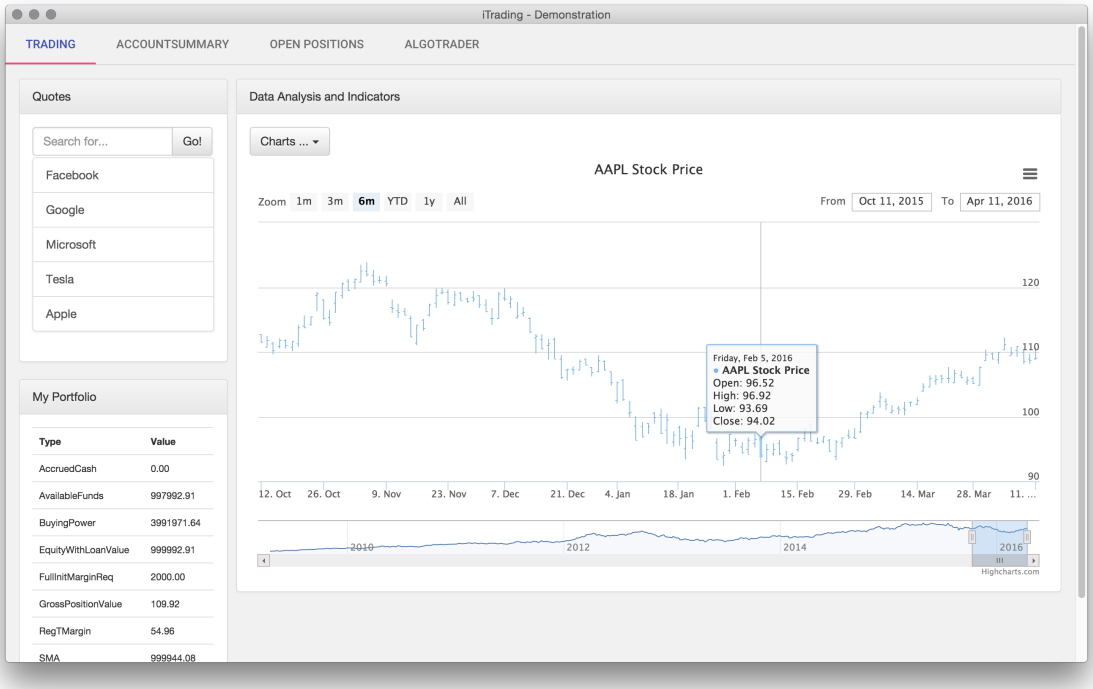


Figure 5.22: iTrading - Trading Window - OHLC Graphic

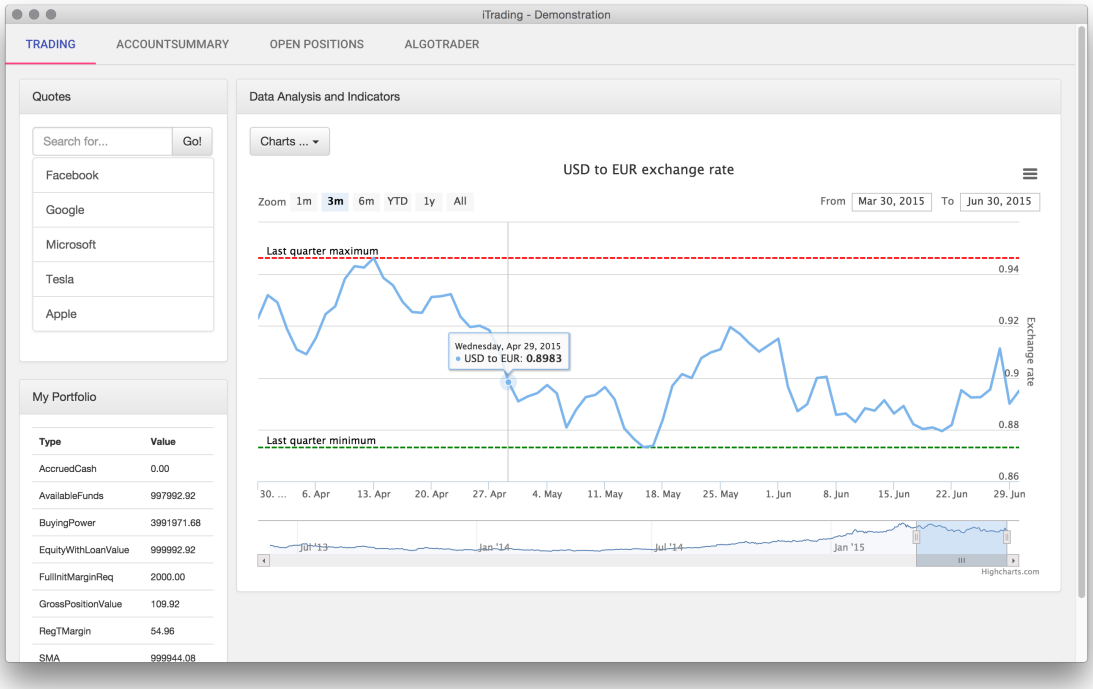


Figure 5.23: iTrading - Trading Window - Last Quarter Graphic

Table 5.1: Trading Window Use Cases

| **Use Case** | **Description** |
|---|---|
| *Analyse Quotes and Indicators* | Ability to have a graphic-centric screen where a trader can visualize and analyse financial markets using many graphic options. |
| *Check Portfolio* | Capacity to observe the trader positions. |
| *Update Data* | Action made by the broker that is constantly updating its data to the client. |
| *Search Quotes* | Search for data to analyse. |

### 5.2.2 Account Summary Window

The main use of Account Summary Window is to show the account summary of the trader. Many types of informations are available such as AccruedCash, Available Funds, Buying Power, among others.

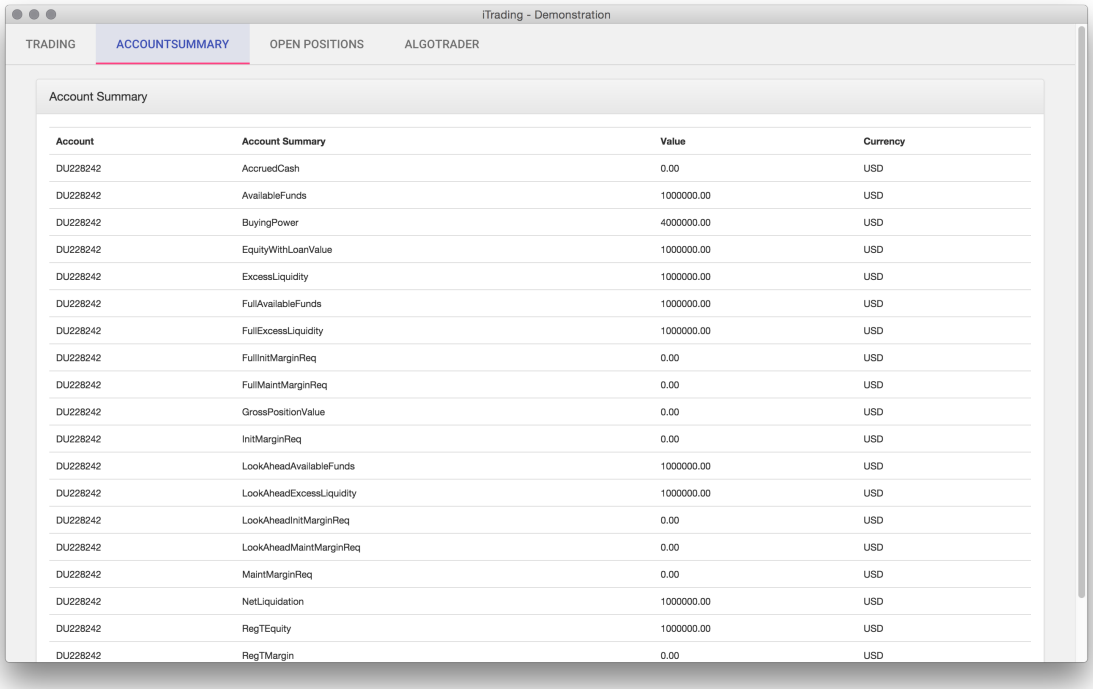


Figure 5.24: iTrading - Account Summary Window

Table 5.2: Account Summary Window Use Cases

| **Use Case** | **Description** |
|---|---|
| *Analyse Account Summary* | The screen on which the user can look in detail to his positions to see his gains and losses. |
| *Update Data* | Action made by the broker who is constantly updating its data to the client. |

### 5.2.3 Open Positions Window

This window enables the trader to analyse each one of the users that were sent from Itrading to the Financial Broker.

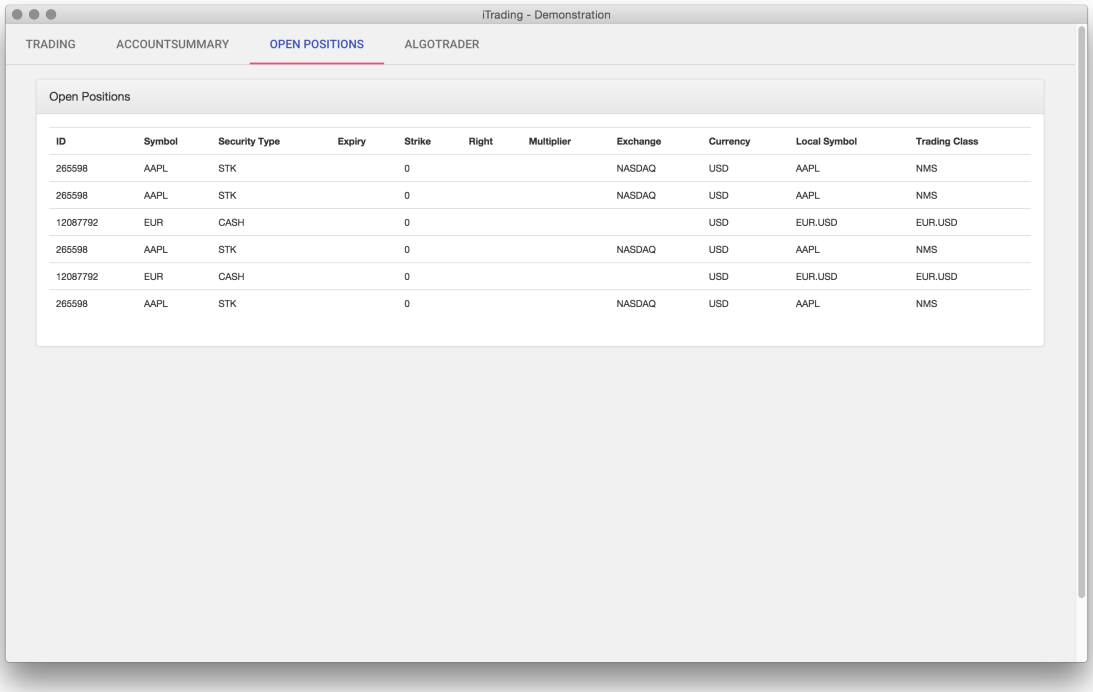


Figure 5.25: iTrading - Open Positions Window

Table 5.3: Open Positions Window Use Cases

| **Use Case** | **Description** |
|---|---|
| *Analyse Orders* | Area where a trader can analyse all the orders that were emitted to the broker. |
| *Update Data* | Action made by the broker that is constantly updating its data to the client. |

### 5.2.4 Algorithmic Trading & Backtesting Window

AlgoTrader window comprises many functionalities of iTrading such as Algorithmic Trading, Backtesting, Automatic Trading, Save/Load Algorithmic Trading scripts, among others.

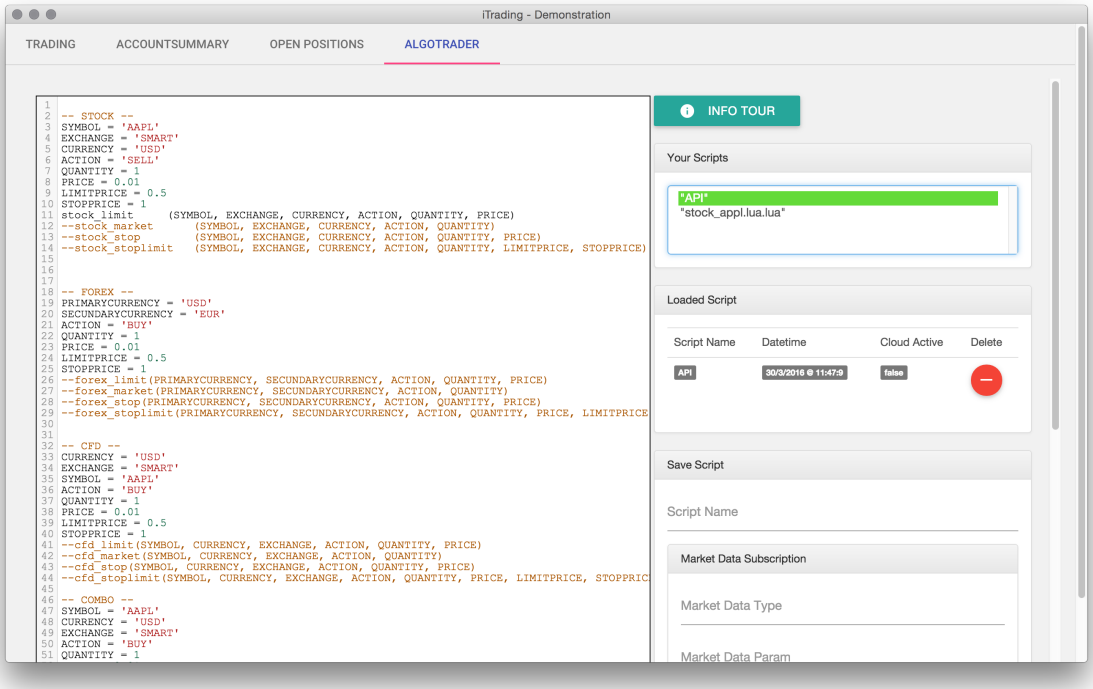


Figure 5.26: iTrading - AlgoTrading Window

In AlgoTrading screen there is a button named *Info Tour*. In pratical terms, it is a documentation shortcut. There, a trader can understand all the windows sections such as IDE, saved scripts, script details, save and automate a script, backtesting and run. Each of this sections have a detailed explanation of how to work with them.

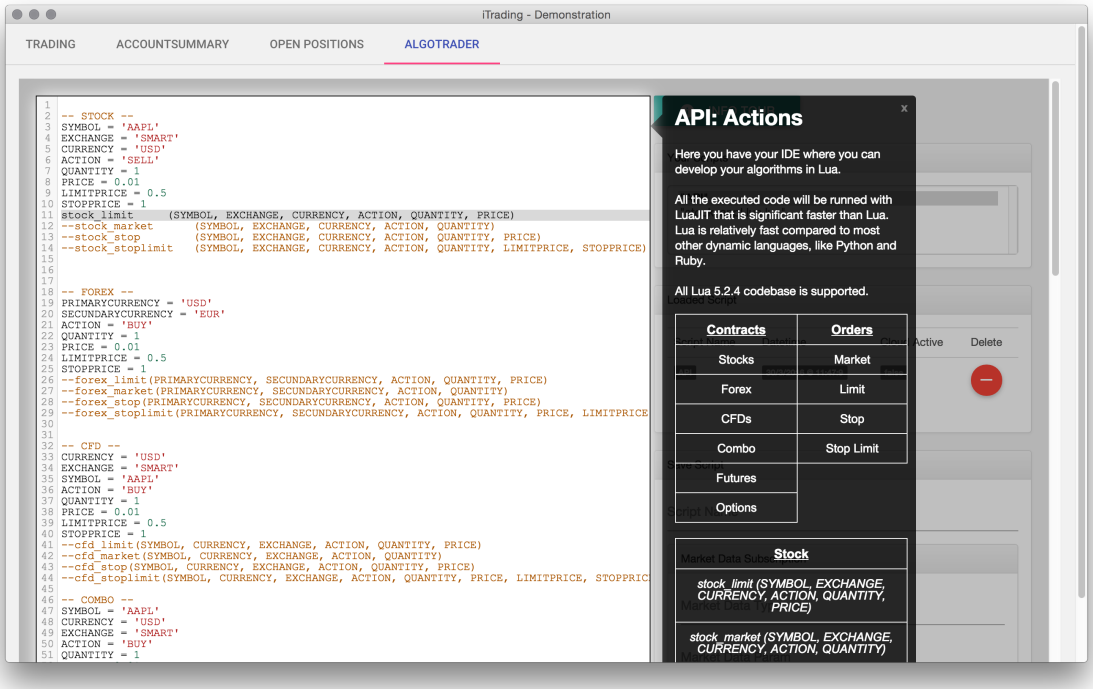


Figure 5.27: iTrading - AlgoTrading Window - API Actions

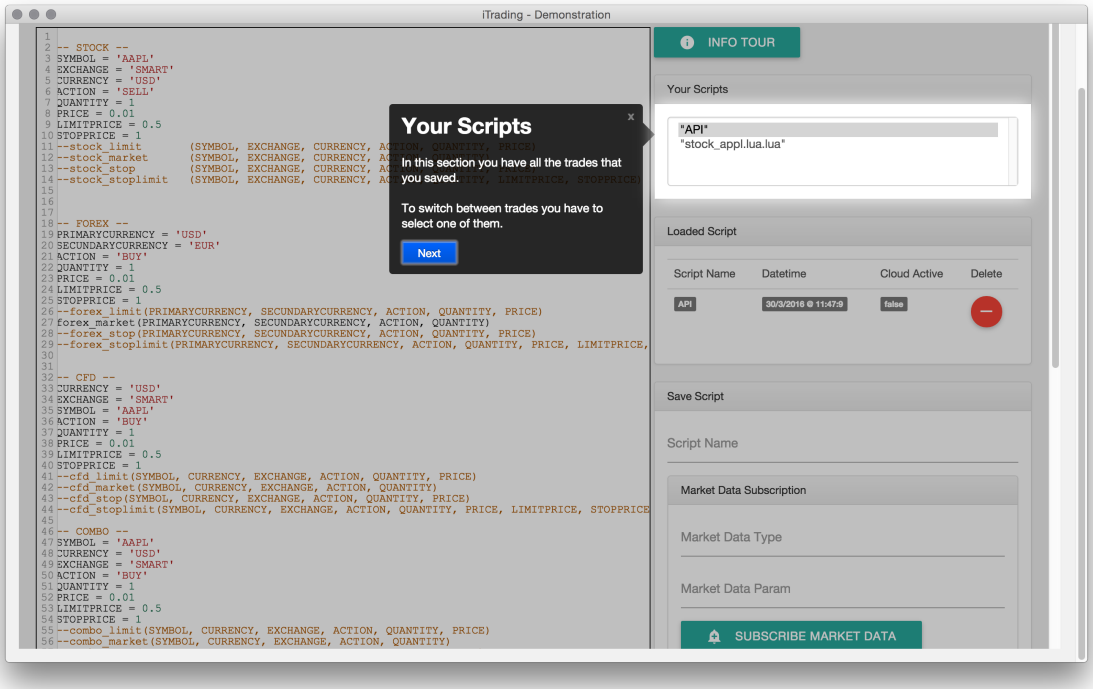


Figure 5.28: iTrading - AlgoTrading Window - Scripts

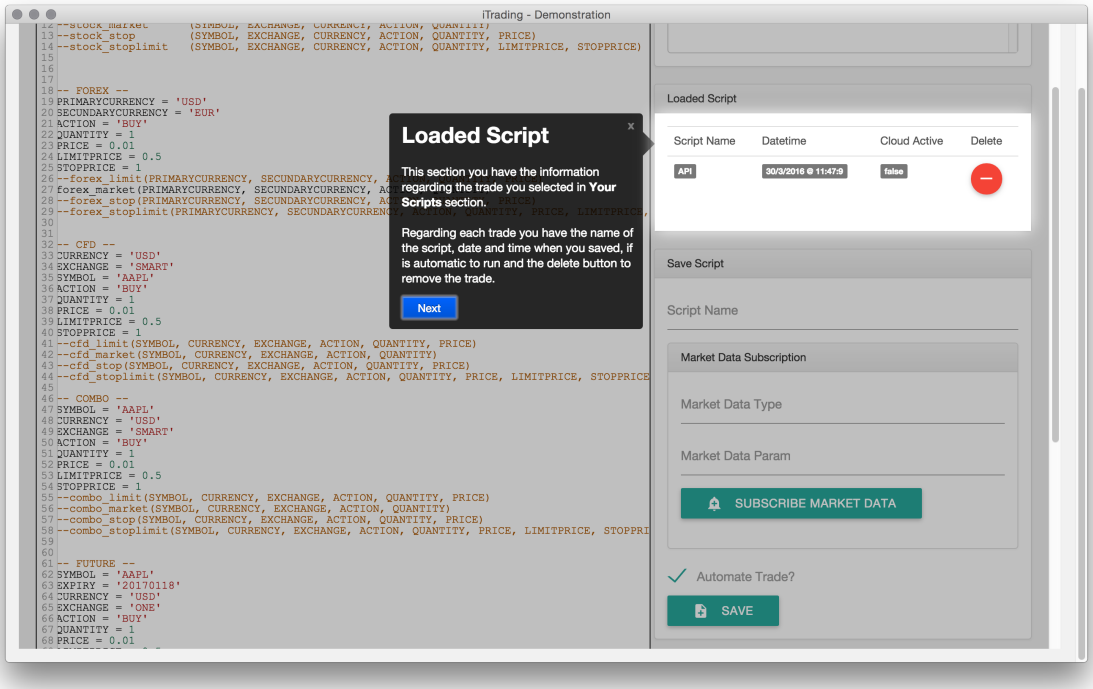


Figure 5.29: iTrading - AlgoTrading Window - Load Scripts

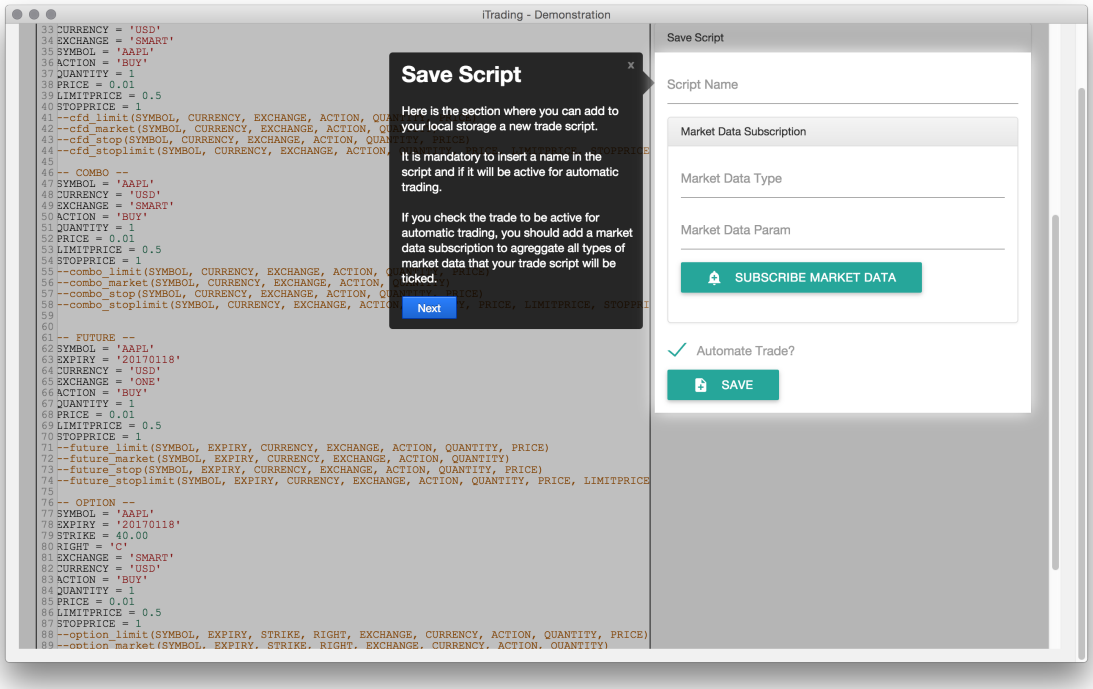


Figure 5.30: iTrading - AlgoTrading Window - Save Scripts

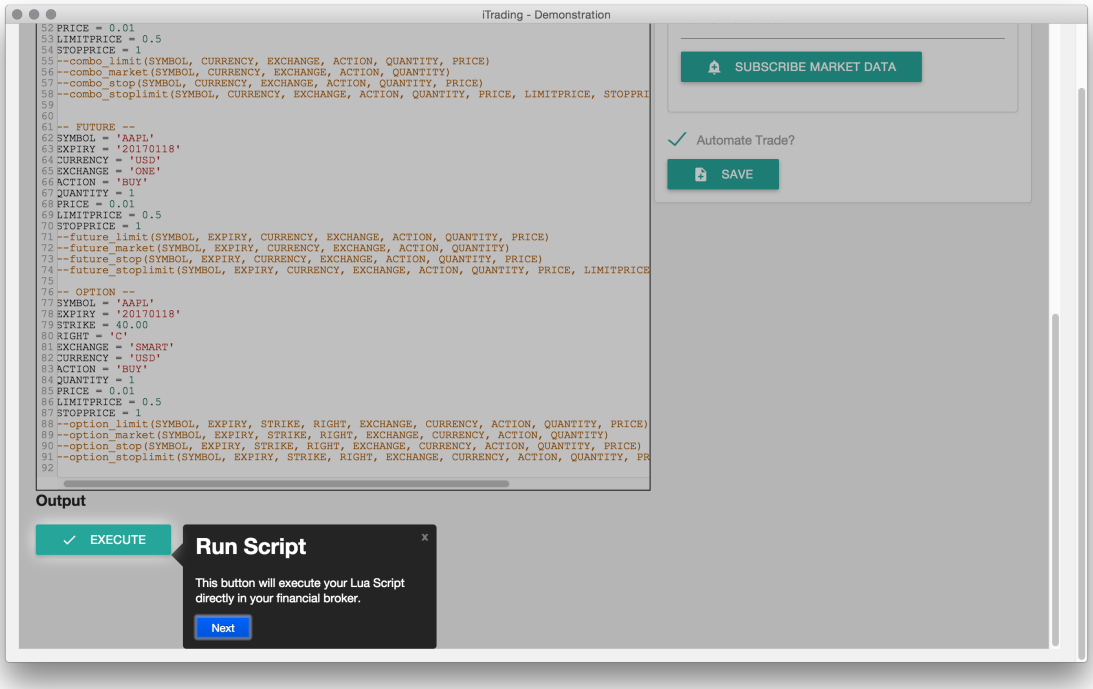


Figure 5.31: iTrading - AlgoTrading Window - Run Scripts

Table 5.4: Algorithmic Trading & Backtesting Window Use Cases

| **Use Case** | **Description** |
|---|---|
| *Place Order* | Place an order on the brokerage company using a broker API. |
| *Save Trading Script* | Option where the trader can, after writing his script, save and decide if he wants to put it running in automating mode. |
| *Listening Ticks* | Broker is always feeding with all the changes that occur in a certain product. This use case will enable trades to run automatically. |
| *Write Trading Script* | System should have an IDE where a trader could write its scripts. |
| *RegMktData* | Financial broker will feed the client application with market data. |
| *Backtesting* | After the execution of a trade, the trader can backtest his algorithm using historical data. |
| *Run Automatically Trading Script* | Run all the scripts defined to run "Automatically". |

### 5.2.5 IB TWS

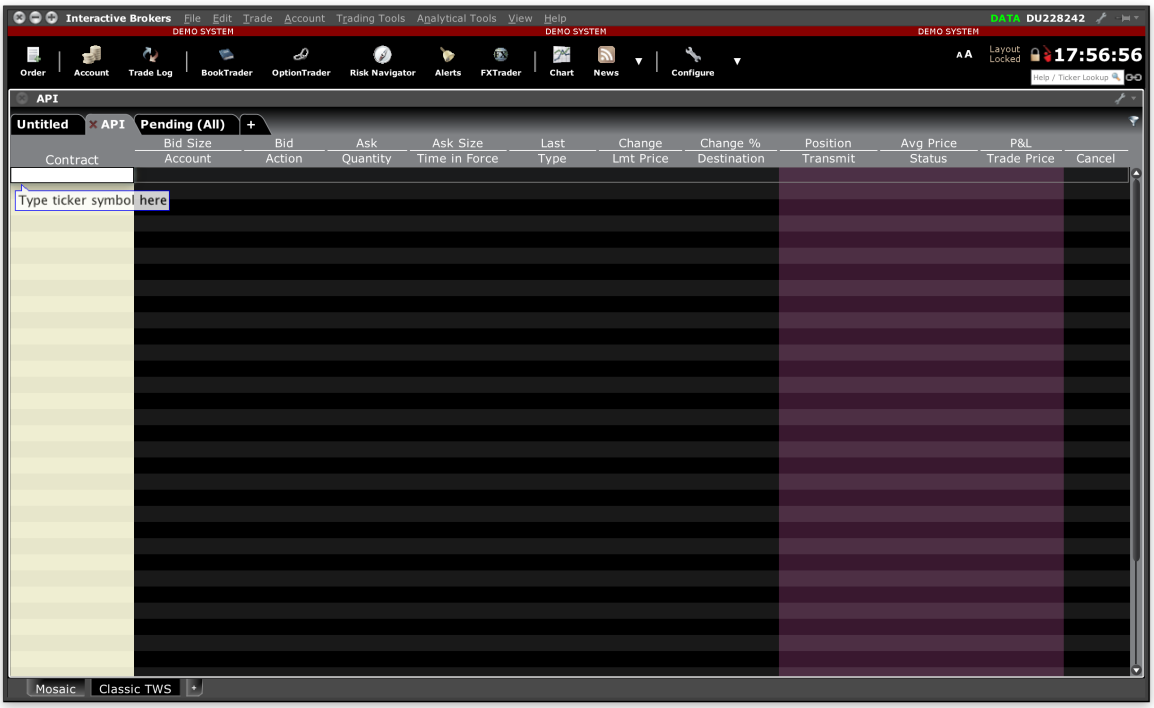


Figure 5.32: Interactive Brokers TWS

Figure 5.33 and 5.34 shows the trades that were made in iTrading.

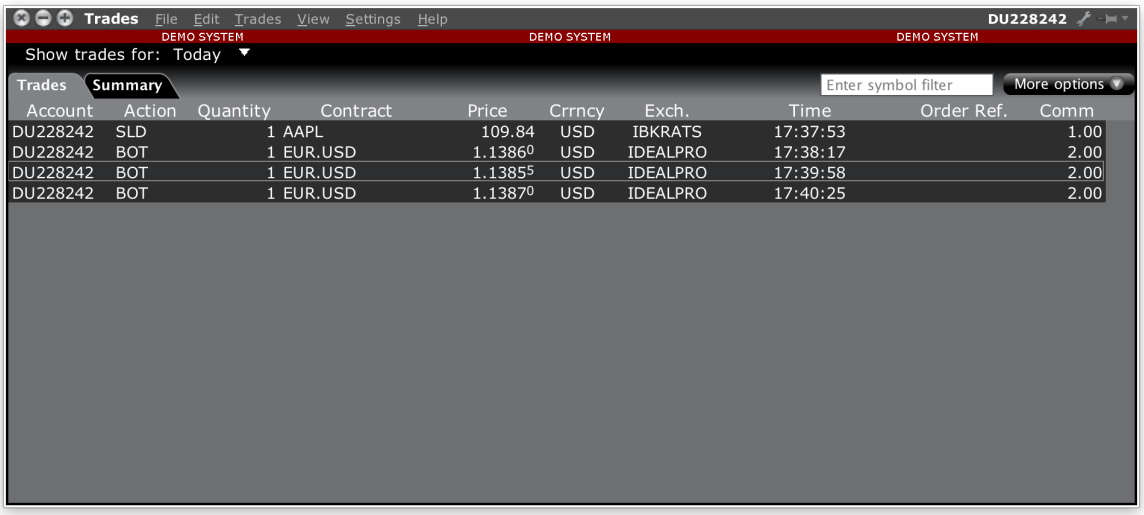


Figure 5.33: Interactive Brokers TWS Trades

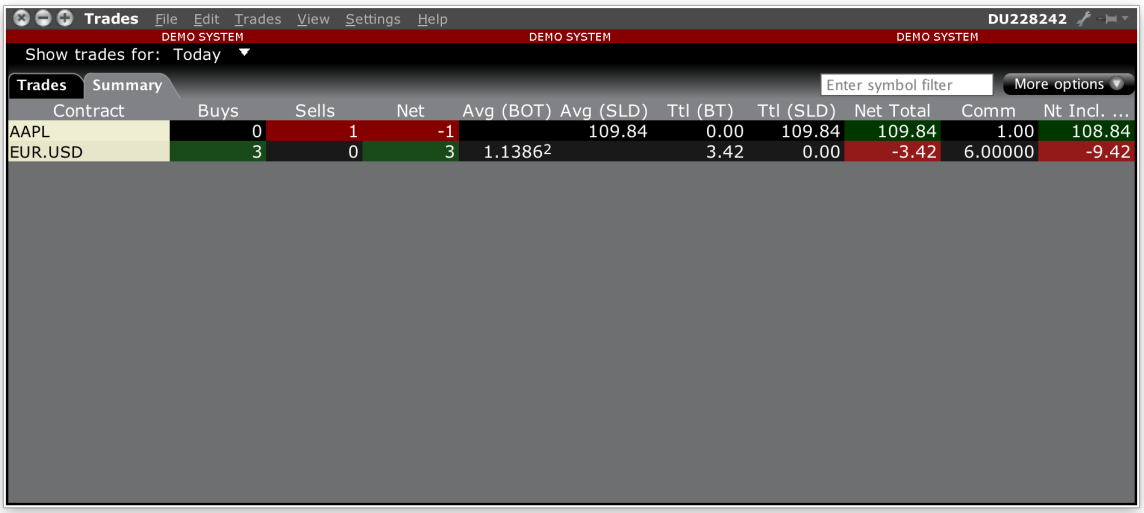


Figure 5.34: Interactive Brokers TWS Summary

IB TWS was very important in this work. As a result, it was possible to confirm if each operation that was made in iTrading is mirrored in the broker too. Figure 5.32 illustrates the main screen of IB TWS API.

## 5.3 Profiling

In software engineering, profiling is a method which dynamically analyse some measures such as the space (memory) or time complexity of a program, the usage of special instructions, or the number and duration of function calls. Most generally, profiling information serves to aid program optimization. To analyse iTrading prototype it were made several examinations regarding the time of Lua scripts execution, script basic operations, getting quotes or account information latency. It was made an analysis of the memory and the setup installation size on all platforms.

Latency is a time period between the stimulation and response, or, from a more generic point of view, a time lag between the cause and the effect of some real change in the system being observed. To analyse latencies of the iTrading prototype, it were made 10 tests of each event.

### Script Execution Latency

It were measured the latency times of Lua execution script (figure 5.35). To do that it were performed four types of executions. Execution without any IB order, with just a hello world, had an average latency time of 17 ms. The value is so low that can't be seen in figure 5.35. It were also performed more three types of execution, with one IB order, with two IB orders and with three IB orders. In fact, this last three events were very similar due the fact of non-blocking HTTP requests made by the event-queue of Node.JS. The average latency of a script with IB orders in it was 5175 ms.

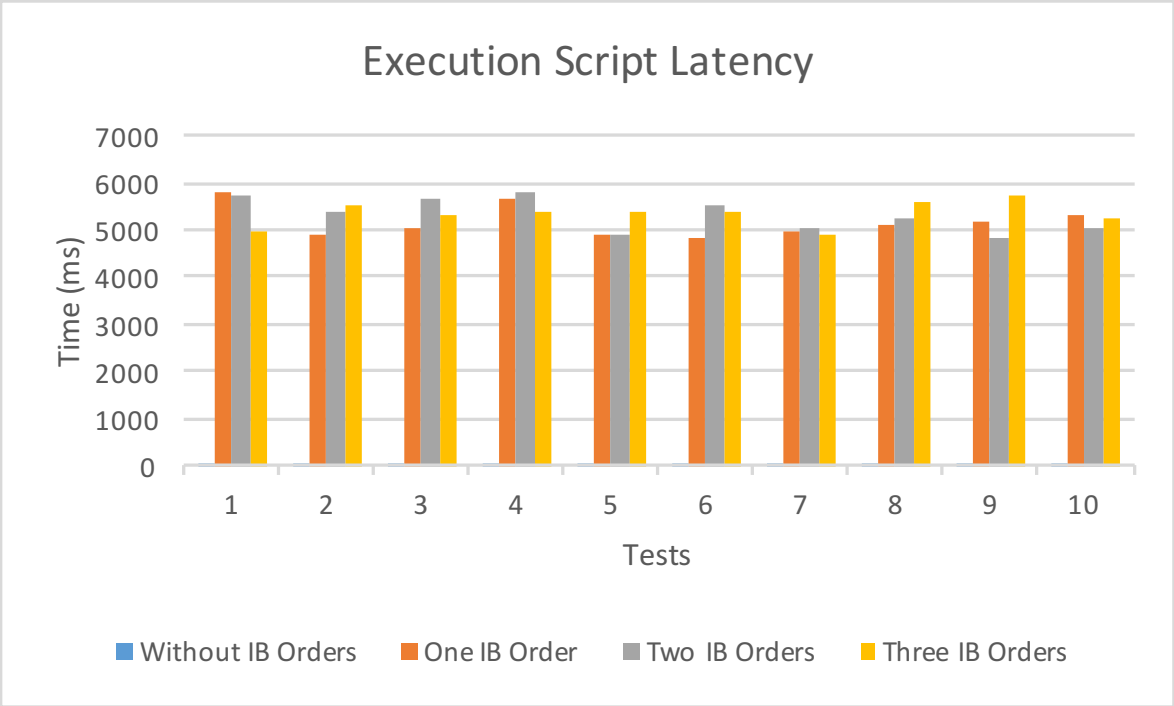


Figure 5.35: Latency Script Execution

### Script Actions Latency

It was also measured the time to save, list and load Lua scripts. The average time of that was 39 ms, 60 ms, and 83 ms, respectively (figure 5.36).

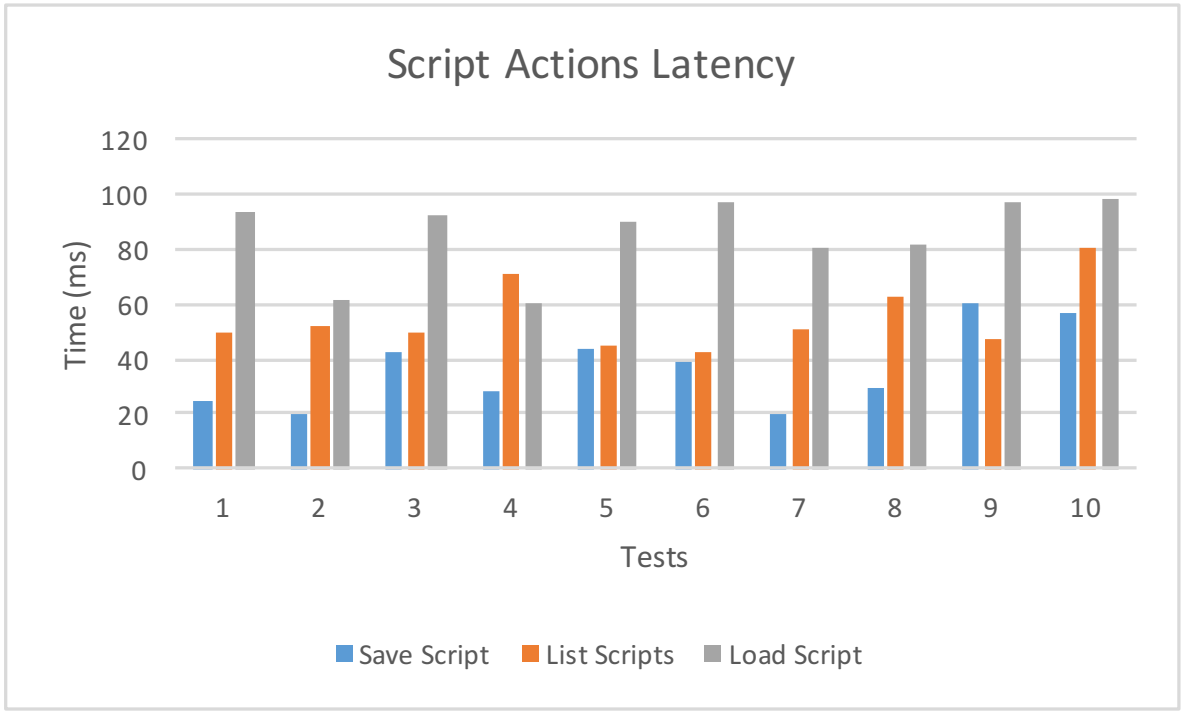


Figure 5.36: Latency Script Actions

### Account Information Latency

Regarding the latency to get quotes to trading screen, request portfolio, request account summary, and request orders the average was 962 ms, 391 ms, 472 ms and 457 ms, respectively (figure 5.37).

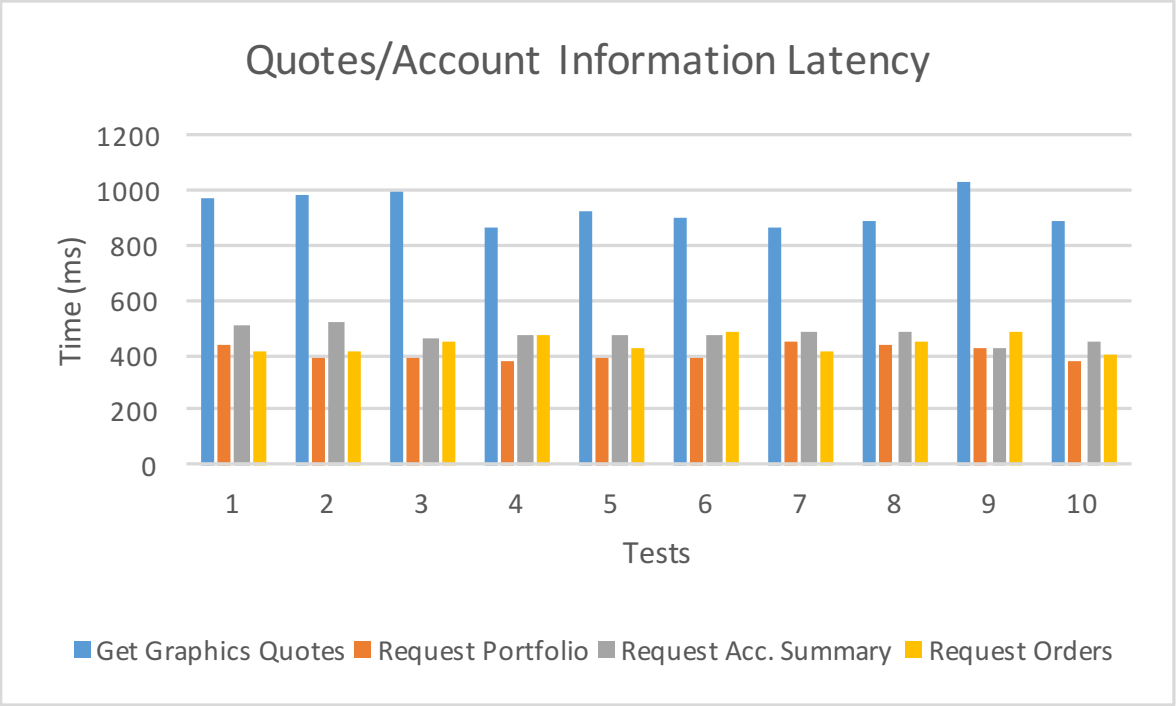


Figure 5.37: Latency Quotes and Account Information

**Memory Used**

The table 5.5 shows the size (MB) occupied in memory for iTrading.

| Platform | Linux | OSX | Windows |
|---|---|---|---|
| Size (MB) | 495 | 270 | 304 |

Table 5.5: Memory used at different platforms

**Size**

The table 5.6 shows the size (MB) of each installation package for all different platforms.

| Platform | Linux | OSX | Windows |
|---|---|---|---|
| Size (MB) | 305 | 142 | 152 |

Table 5.6: Compressed iTrading Setups for different platforms

## 5.4 Chapter Summary

In order to evaluate the results, it was written an user interface questionnaire. It aimed to find usability problems. The questionnaire was answered by 23 different people. Many principles were examined such as: visibility of system status; error prevention; user control and

freedom, among others. The results of the questionnaire were very positive. Many participants said that the system speaks the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. On the other hand, some answers said that there was an error message that it is not expressed in plain language.

Furthermore, it were explored many latencies namely: list graphics; list account summary/orders; execute script; save script and set it to run automatically. In particular, the most relevant latency is the time to run a script. The time delay between the action of running a script and receiving its response was in average 17 milliseconds (ms). Identically, it was measured the time between the action of running a script, now with financial orders, and receiving its response. The latency was in average 5175 ms.

# 6 Conclusion

In this chapter, it is described the contributions that were made during this dissertation and the future work.

## 6.1 Future Work

The future work can take many directions in every module that was developed. Starting in the effective trading, it can be developed a manual interface. Many traders aren't familiar with Lua programming language or didn't have any programming skills so they prefer to introduce their orders manually. The API developed and the work to be done is all about the interface development. Another programming language that can be used in the integrated development environment is Python, taking advantage of its finance powerful libraries. Another possible feature is to create a new module capable of alerting the trader, via email ou SMS, if some kind of fluctuation occurs.

New modules can be developed too. A chat room would be an advantage in a system like this, where users may share ideas on developing their strategies. In addition, creating more indicators will be an advantage for analyzing financial products. A good feature of such trading software is a GUI with components moveable and resizable. This will produce gains in trading productivity if the trader is comfortable with its layout. Furthermore, a panel with recent news will have a positive impact since the trader could be aware of economy news and act accordingly on the same software.

Lastly, another possible feature is migrating iTrading to mobile platforms. All the logic code is separated - Browser-side and Renderer-side - so developing a mobile application would be easy. However, it will require a server to operate with Node.JS/IB. This could mean privacy issues since the traders may not want to share their trading scripts to servers.

## 6.2 Final consideration

All things considered, there were many positive developments in the subject. It was used in algorithmic trading a different scripting language, Lua, due to its many advantages when comparing with its alternatives. Lua is portable, fast and multi-paradigm. Besides, it is one

of the easiest scripting languages to learn. Moreover, it was created a trading application which will run on every platform, thus differentiating it from the existing softwares available in the market. It were used cutting-edge technologies such as Electron.JS and Node.JS. These technologies enabled the creation of different trading softwares with features such as interactive graphics. All the system was build to be easy to decouple. As a result, it will be more easy in a future development of a web or a mobile solution. Finally, it were created many trading functionalities such as indicators, backtesting and algorithmic trading.

On the other hand, there were less positive points also. Since it is a hybrid application, it isn't completely native. Furthermore, it were used many APIs that aren't free. Consequently, the use of them is limited, for instance, data sources for graphics visualization it's different from the ones used in trading screen. In addition, the number of testers, and the inexperience of them in such trading softwares, may compromise the evaluation and results of the application.

## 6.3 Contributions

The work developed in this dissertation contributed to a new kind of trading platform. It was developed a set of features such as algorithmic trading in Lua, embedded IDE, cross-platform, backtesting, interactive graphics, automatic trading, among others which make it the first application of its type. Additionally, it was studied the development of a low-latency application using a JavaScript cross-platform technology based on Node.JS. On balance, with Electron.JS it was possible to build a performant, data-driven desktop trading application. As a result, the technology used was adequate. Besides, it was submitted an article based on this work to IT Symposium, INFORUM [84].

# Bibliography


[1] Stephen McBride. States against markets: The limits of globalization robert boyer and daniel drache, eds. london: Routledge, 1996, pp. xii, 448. *Canadian Journal of Political Science*, 30:776–777, 12 1997.

[2] Economy Watch. Financial market growth. Web. Published November 2010. [http://www.economywatch.com/market/financial-market/growth.html].

[3] Laura Alfaro, Areendam Chanda, Sebnem Kalemli-Ozcan, and Selin Sayek. FDI and economic growth: The role of local financial markets. *Journal of International Economics*, 64(1):89–112, 2004.

[4] Nicholas Economides. "the impact of the internet on financial markets". *Journal of Financial Transformation*, 1(1):8–13, 2001.

[5] Keiko Kubota Helen V. Milner. Why the move to free trade? democracy and trade policy in the developing countries. *International Organization*, 59(1):107–143, 2005.

[6] CHARLES M. JONES TERRENCE HENDERSHOTT and ALBERT J. MENKVELD. "does algorithmic trading improve liquidity?". *THE JOURNAL OF FINANCE*, LXVI,(1):8–13, 2011.

[7] M. A H Dempster, T. W. Payne, Y. Romahi, and G. W P Thompson. Computational learning techniques for intraday FX trading using popular technical indicators. *IEEE Transactions on Neural Networks*, 12(4):744–754, 2001.

[8] William J O Neil. 24 Essential Lessons for Investment Success. *Most*, 6, 2000.

[9] Alain Chaboud. Rise of the Machines : Algorithmic Trading in the Foreign Exchange Market. 2012.

[10] Bruno Biais and Sophie Moinas. Equilibrium High Frequency Trading 1. 2011.

[11] Thierry Foucault, Johan Hombert, and Ioanid Rosu. News Trading and Speed. pages 1–53, 2012.

[12] Robert Jarrow, Robert A Jarrow, and Philip Protter. A Dysfunctional Role of High Frequency Trading in Electronic Markets This paper can be downloaded without charge at The Social Science Research Network Electronic Paper Collection . A Dysfunctional Role of High Frequency Trading in Electronic Markets. (March), 2011.

[13] Jeremy C Stein. Presidential Address : Sophisticated Investors and Market Efficiency. LXIV(4), 2009.

[14] Joel Hasbrouck and Gideon Saar. Low-latency trading $. *Journal of Financial Markets*, 16(4):646–679, 2013.

[15] Wiley Trading. *E r n e s t p. c h a n.*

[16] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes. Lua Programming Language. 2011.

[17] Roberto Ierusalimschy, Luiz Henrique De Figueiredo, and Waldemar Celes. Lua 5 . 2 Reference Manual 1 - Introduction 2 - Basic Concepts. pages 1–133, 2014.

[18] LuaWebSite. about @ www.lua.org, 2016.

[19] Shayne Fletcher and Christopher Gardner. Financial Modelling in Python. page 244, 2010.

[20] Hans Petter Langtangen. *Python Scripting for Computational Science*, volume 3. 2008.

[21] MQL4 Reference. index @ docs.mql4.com.

[22] MetaQuotes Software Corp. Metatrader. Web 20/7/15 [http://www.metatrader4.com].

[23] mt5bot. Programming of mql4 vs mql5. Web 20/7/15 [http://mt5bot.blogspot.pt/2011/01/programming-of-mql4-vs-mql5.html].

[24] Metatrader. Types of execution. Web 20/7/15 [www.metatrader5.com/en/terminal/help/trading/general_concept/execution_types].

[25] Wikipedia. Market depth. Web 20/7/15 [https://en.wikipedia.org/wiki/Market_depth].

[26] MetaQuotes. Metatrader 4 trading platform. Web 28/10/15 [[http://www.metaquotes.net].

[27] Jafar Calley. Installing metatrader 4 forex charts under linux. Web 22/7/15 [aboutcurrency.com/university/metatrader/installing_metatrader4_under_linux.shtml].

[28] Metatrader. How to install metatrader on mac. Web 22/7/15 [http://www.metatradermac.info].

[29] Metatrader. Installing wine on mac os. Web 22/7/15 [https://www.mql5.com/en/articles/1356].

[30] Protrader. Protrader desktop platform. Web 22/7/15 [http://protrader.com].

[31] Protrader. Protrader web platform. Web 22/7/15 [http://protrader.com].

[32] ProTrader. Protrader server. Web 28/10/15 [http://protrader.com/platform/server].

[33] MetaStock. Metastock. Web 05/11/15 [[http://metastock.com].

[34] TC2000. Tc2000. Web 10/12/15 [[http://www.worden.com].

[35] eSignal. esignal. Web 10/12/15 [[http://www.esignal.com].

[36] NinjaTrader. Ninjatrader. Web 10/12/15 [http://ninjatrader.com/].

[37] W59. W59 pro. Web 05/11/15 [http://www.wave59.com].

[38] EquityFeed. Equityfeed. Web 10/12/15 [http://equityfeed.com/].

[39] ProfitSource. Profitsource. Web 10/12/15 [http://www.profitsource.com/].

[40] VectorVest. Vectorvest. Web 10/12/15 [http://www.vectorvest.com].

[41] MarketClub. Marketclub. Web 10/12/15 [http://club.ino.com].

[42] Shobhit Seth. The best technical analysis trading software. Web 28/10/15 [http://www.investopedia.com].

[43] Dictionary.com. Broker definition. Web 14/03/15 [http://www.dictionary.com].

[44] Investopedia.com. Broker definition. Web 14/03/15 [http://www.investopedia.com].

[45] Investopedia.com. Picking first broker. Web 10/12/15 [http://www.investopedia.com].

[46] Interactive Broker. Interactive broker. Web 3/05/16 [https://www.interactivebrokers.com].

[47] Eric Roberts and Antoine Picard. Designing a Java graphics library for CS 1. *ACM SIGCSE Bulletin*, 30(3):213–218, 1998.

[48] Qt. Qt. Web 26/12/15 [http://www.qt.io].

[49] sabioguru. Let's start with the qt: First story 'write once, compile anywhere' fantastic toolkit, qt. Web 26/12/15 [http://blog.qt.io/blog/2011/01/26/startqt-write-once-compile-anywhere-qt/].

[50] QCustomPlot. Qcustomplot. Web 27/12/15 [http://www.qcustomplot.com/].

[51] Chromium Embedded Framework. Chromium embedded framework. Web 10/12/15 [http://maxogden.com/electron-fundamentals.htmlhttps://bitbucket.org/chromiumembedded/cef].

[52] Horia Olaru. The chromium embedded framework. Web 26/12/15 [http://blogs.adobe.com/webplatform/2013/05/01/the-chromium-embedded-framework/].

[53] JavaScripts vs Java. Javascripts vs java. Web 27/12/15 [http://benchmarksgame.alioth.debian.org/u64q/javascript.html].

[54] JavaScripts vs C++. Javascripts vs c++. Web 27/12/15 [http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=v8lang2=gpp].

[55] R. Cecco. *Supercharged JavaScript Graphics: With HTML5 Canvas, JQuery, and More*. O'Reilly Series. O'Reilly Media, Incorporated, 2011.

[56] Nick Qi Zhu. *Data Visualization with D3.JS Cookbook*. 2013.

[57] Max Ogden. Electron fundamentals. Web 10/12/15 [http://maxogden.com].

[58] Chromium Community. Google chromium content module. Web 27/12/15 [http://www.chromium.org/developers/content-module].

[59] NW.js Community. nwjs/chromium.src. Web 27/12/15 [https://github.com/nwjs/chromium.src].

[60] Microsoft. Visual studio code editor. Web 27/12/15 [https://code.visualstudio.com].

[61] Facebook. Nuclide editor. Web 27/12/15 [http://nuclide.io].

[62] maxogden. Screencat. Web 27/12/15 [https://github.com/maxogden/screencat].

[63] moose team. Friends. Web 27/12/15 [https://github.com/moose-team/friends].

[64] IO.js Community. Io.js. Web 27/12/15 [https://iojs.org].

[65] Google Open-Source Community. Chromium. Web 27/12/15 [http://www.chromium.org].

[66] Kendall Scott, Publisher Addison Wesley, Wesley Longman, and Massachusetts Harlow. *UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language Martin Fowler UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language UML Distilled Second Edition A Brief Guide to the Standard Object Modelli*. 1999.

[67] ilyavorobiev. Electron architecture. Web 27/12/15 [https://github.com/ilyavorobiev/atom-docs/blob/master/atom-shell/Architecture.md].

[68] NodeJS Community. Nodejs. Web 3/05/16 [https://nodejs.org].

[69] libuv. libuv foundation. Web 14/04/16 [http://docs.libuv.org].

[70] Interactive Broker. Ib java api. Web 3/05/16 [https://www.interactivebrokers.com/en/software/api/apiguide/java/java.htm].

[71] NodeJS Community. Nodejs net module. Web 3/05/16 [https://nodejs.org/api/net.html].

[72] NodeJS Community. Nodejs event emitter. Web 3/05/16 [https://nodejs.org/api/events.html].

[73] kripten. lua.vm.js. Web 27/04/16 [https://kripken.github.io/lua.vm.js].

[74] Lua. Lua. Web 27/04/16 [http://www.lua.org/].

[75] kripten. emscripten. Web 27/04/16 [http://kripken.github.io/emscripten-site/].

[76] arewefastyet. arewefastyet. Web 27/04/16 [http://arewefastyet.com].

[77] socket.io. socket.io. Web 3/05/16 [socket.io].

[78] Quandl. Quandl. Web 18/04/16 [quandl.com].

[79] Francis Smart. quandl_formats. Web 18/04/16 [http://www.econometricsbysimulation.com].

[80] codemirror. codemirror. Web 18/04/16 [http://codemirror.net/].

[81] Google AngularJS. What is angular? Web 2/05/16 [https://docs.angularjs.org].

[82] Google. Material design. Web 19/04/16 [https://design.google.com/].

[83] Jacob Neilsen, Heuristic Analysis, Introduction Jakob Nielsen, and Introduction Heuristics. M4 L4 Nielsen ' s Ten Heuristics. pages 1–6, 1994.

[84] Inforum. INForum2016 @ inforum.org.pt.

# Appendices

# A Code Snippets

```
// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow;

function createWindow () {

  // Create the browser window.
  mainWindow = new BrowserWindow({width: 1920, height: 1080, title: "iTrading -
      Demonstration"});

  // and load the index.html of the app.
  mainWindow.loadURL('file://' + __dirname + '/index.html');

  // Open the DevTools.
  //mainWindow.webContents.openDevTools();

  // Emitted when the window is closed.
  mainWindow.on('closed', function() {
    // Dereference the window object, usually you would store windows
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    mainWindow = null;
  });
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
app.on('ready', createWindow);


// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
app.on('ready', function() {

    // In main process.
    const ipcMain = require('electron').ipcMain;
```

```
    ipcMain.on('asynchronous-message', function(event, arg) {
    console.log(arg); // prints "ping"
    event.sender.send('asynchronous-reply', 'tick');
    });

    ...
    // Intantiate Automatic Trading Process
    ...
});


// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On OS X it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', function () {
  // On OS X it's common to re-create a window in the app when the
  // dock icon is clicked and there are no other windows open.
  if (mainWindow === null) {
    createWindow();
  }
});
```

Listing A.1: Control Lifecycle of the iTrading.

```
// In browser.
var ipc = require('ipc');
ipc.on('asynchronous-message', function(event, arg) {
  console.log(arg); // prints "ping"
  event.sender.send('asynchronous-reply', 'pong');
});

ipc.on('synchronous-message', function(event, arg) {
  console.log(arg); // prints "ping"
  event.returnValue = 'pong';
});

// In web page.
var ipc = require('ipc');
console.log(ipc.sendSync('synchronous-message', 'ping')); // prints "pong"

ipc.on('asynchronous-reply', function(arg) {
  console.log(arg); // prints "pong"
});
ipc.send('asynchronous-message', 'ping');
```

Listing A.2: Communication between Browser and Renderer via IPC.

```
var remote = require('remote');
var BrowserWindow = remote.require('browser-window');
var win = new BrowserWindow({ width: 800, height: 600 });
win.loadUrl('https://joelpinheiro.github.io/itrading');
```

Listing A.3: Communication between Browser and Renderer via Remote.

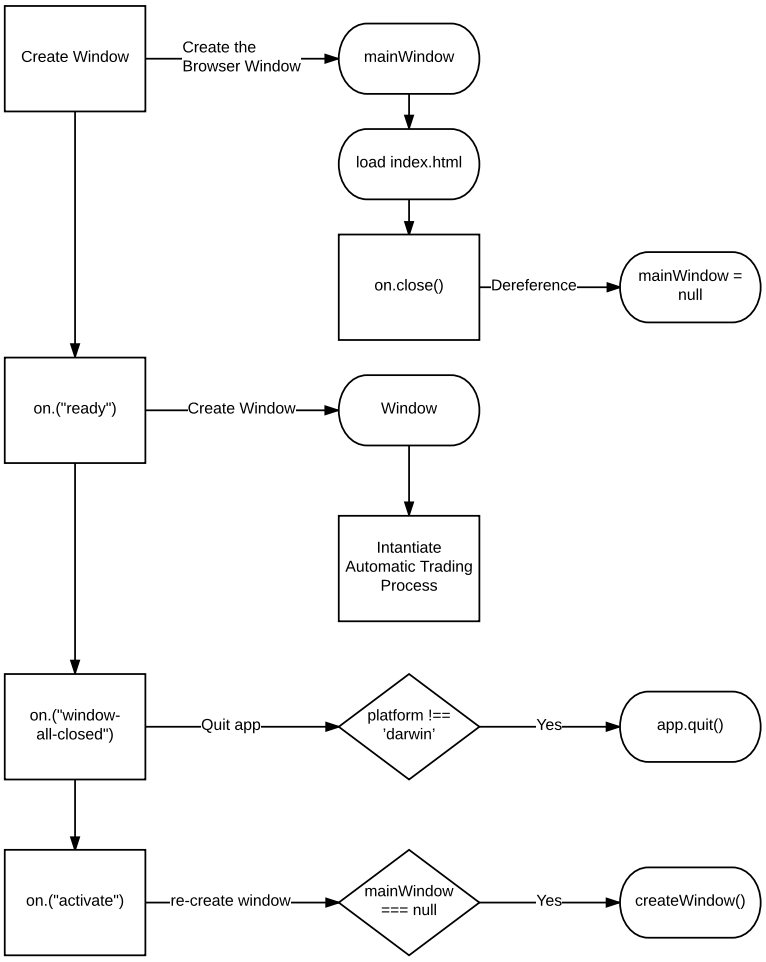


Figure A.1: Electron Lifecycle

```
[
   {
      "script_name":"\"123.lua\"",
      "lua_script":"\"\\n-- STOCK --\\nSYMBOL = 'AAPL'\\nEXCHANGE =
          'SMART'\\nCURRENCY (...)\"",
      "cloud_active":"\"true\"",
      "datetime":"\"3/6/2016 @ 14:55:32\"",
      "marketdata_list":"\"[]\""
   }
]
```

| Stock Contract |
|---|
| stock_limit (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, PRICE) |
| stock_market (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY) |
| stock_stop (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, PRICE) |
| stock_stoplimit (SYMBOL, EXCHANGE, CURRENCY, ACTION , QUANTITY, LIMITPRICE, STOPPRICE) |

Table A.1: Stock Contract - IB API's Methods

| Forex Contract |
|---|
| forex_limit(PRIMARYCURRENCY, SECUNDARYCURRENCY , ACTION, QUANTITY, PRICE) |
| forex_market(PRIMARYCURRENCY, SECUNDARYCURRENCY , ACTION, QUANTITY) |
| forex_stop(PRIMARYCURRENCY, SECUNDARYCURRENCY , ACTION, QUANTITY, PRICE) |
| forex_stoplimit(PRIMARYCURRENCY, SECUNDARYCURRENCY , ACTION, QUANTITY, PRICE, LIMITPRICE, STOPPRICE) |

Table A.2: Forex Contract - IB API's Methods

| CFDs Contract |
|---|
| cfd_limit(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE) |
| cfd_market(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY) |
| cfd_stop(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE) |
| cfd_stoplimit(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE, LIMITPRICE, STOPPRICE) |

Table A.3: CFDs Contract - IB API's Methods

Listing A.4: Stored script in JSON

| Combo Contract |
| --- |
| combo_limit(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE) |
| combo_market(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY) |
| combo_stop(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE) |
| combo_stoplimit(SYMBOL, CURRENCY, EXCHANGE, ACTION , QUANTITY, PRICE, LIMITPRICE, STOPPRICE) |

Table A.4: Combo Contract - IB API's Methods

| Option Contract |
| --- |
| option_limit(SYMBOL, EXPIRY, STRIKE, RIGHT, EXCHANGE , CURRENCY, ACTION, QUANTITY, PRICE) |
| option_market(SYMBOL, EXPIRY, STRIKE, RIGHT, EXCHANGE , CURRENCY, ACTION, QUANTITY) |
| option_stop(SYMBOL, EXPIRY, STRIKE, RIGHT, EXCHANGE , CURRENCY, ACTION, QUANTITY, PRICE) |
| option_stoplimit(SYMBOL, EXPIRY, STRIKE, RIGHT, EXCHANGE , CURRENCY, ACTION, QUANTITY, PRICE, LIMITPRICE, STOPPRICE) |

Table A.5: Option Contract - IB API's Methods

| Future Contract |
| --- |
| future_limit(SYMBOL, EXPIRY, CURRENCY, EXCHANGE , ACTION, QUANTITY, PRICE) |
| future_market(SYMBOL, EXPIRY, CURRENCY, EXCHANGE , ACTION, QUANTITY) |
| future_stop(SYMBOL, EXPIRY, CURRENCY, EXCHANGE , ACTION, QUANTITY, PRICE) |
| future_stoplimit(SYMBOL, EXPIRY, CURRENCY, EXCHANGE , ACTION, QUANTITY, PRICE, LIMITPRICE, STOPPRICE) |

Table A.6: Future Contract - IB API's Methods

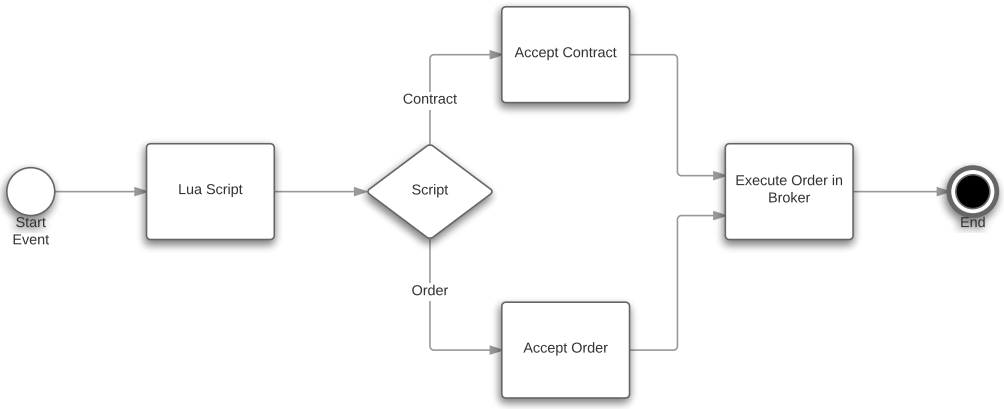


Figure A.2: iTrading - Executing Orders

| | 71<br>% do total: 100,00% (71) |
|---|---|
| 1. https://atnog.av.it.pt/~joelpinheiro/iTrading-win32-x64.zip | **19** (26,76%) |
| 2. https://www.dropbox.com/s/34ifrgntg9ayacq/iTrading-darwin-x64.zip?dl=0 | **15** (21,13%) |
| 3. https://www.dropbox.com/s/o0l35knmm0ej6kn/iTrading-win32-x64.zip?dl=0 | **11** (15,49%) |
| 4. https://atnog.av.it.pt/~joelpinheiro/iTrading-linux-x64.zip | **10** (14,08%) |
| 5. https://atnog.av.it.pt/~joelpinheiro/iTrading-darwin-x64.zip | **8** (11,27%) |
| 6. https://www.dropbox.com/s/sjqr4s3txj5pn2b/iTrading-linux-x64.zip?dl=0 | **8** (11,27%) |

Figure A.3: Google Analytics - Downloads

| | 726<br>% do total: 100,00% (726) | 84,99%<br>Média por visualização de propriedade: 84,99% (0,00%) | 617<br>% do total: 100,00% (617) | 80,58%<br>Média por visualização de propriedade: 80,58% (0,00%) | 1,25<br>Média por visualização de propriedade: 1,25 (0,00%) | 00:01:08<br>Média por visualização de propriedade: 00:01:08 (0,00%) |
|---|---|---|---|---|---|---|
| 1. (direct) / (none) | **310** (42,70%) | 85,48% | 265 (42,95%) | 84,52% | 1,17 | 00:00:49 |
| 2. reddit.com / referral | **257** (35,40%) | 88,33% | 227 (36,79%) | 77,82% | 1,28 | 00:01:09 |
| 3. facebook.com / referral | **32** (4,41%) | 81,25% | 26 (4,21%) | 81,25% | 1,06 | 00:00:11 |
| 4. cookie-law-enforcement-ff.xyz / referral | **30** (4,13%) | 100,00% | 30 (4,86%) | 100,00% | 1,00 | 00:00:00 |
| 5. google / organic | **25** (3,44%) | 44,00% | 11 (1,78%) | 52,00% | 2,20 | 00:04:39 |
| 6. news.ycombinator.com / referral | **21** (2,89%) | 85,71% | 18 (2,92%) | 80,95% | 1,19 | 00:00:09 |
| 7. caldeiraodebolsa.jornaldenegocios.pt / referral | **13** (1,79%) | 69,23% | 9 (1,46%) | 84,62% | 1,15 | 00:00:25 |

Figure A.4: Google Analytics - Source

# B Emscripten

```
var ib = new (require('ib'))({
  // clientId: 0,
  // host: '127.0.0.1',
  // port: 7496
}).on('error', function (err) {
  console.error('error --- %s', err.message);
}).on('result', function (event, args) {
  console.log('%s --- %s', event, JSON.stringify(args));
}).once('nextValidId', function (orderId) {
  ib.placeOrder(
    orderId,
    ib.contract.stock('AAPL'),
    ib.order.limit('BUY', 1, 0.01) // safe, unreal value used
  );
  ib.reqOpenOrders();
}).once('openOrderEnd', function () {
  ib.disconnect();
})

ib.connect()
  .reqIds(1);
```

Listing B.1: NodeIB Usage

Emscripten is an LLVM-based project that compiles C and C++ into highly-optimizable JavaScript in asm.JS format. With this is possible to run C and C++ on the web at a near-native speed, without plugins. Emscripten can convert OpenGL into WebGL making it possible to use familiar APIs like SDL or HTML5 directly. Emscripten is an Open-Source Low Level Virtual Machine (LLVM) to JavaScript compiler. Using Emscripten you can:

- Compile C and C++ code into JavaScript;
- Compile any other code that can be translated into LLVM bitcode into JavaScript;
- Compile the C/C++ runtimes of other languages into JavaScript, and then run code in those other languages in an indirect way (this has been done for Python and Lua).

Emscripten makes native code immediately available on the Web: a platform that is standards-based, has numerous independent compatible implementations, and runs everywhere from PCs to iPads. With Emscripten, C/C++ developers don't have the high cost of porting code manually to JavaScript — or having to learn JavaScript at all. Web developers also benefit, as they can use the many thousands of pre-existing native utilities and libraries in their sites. Practically any portable C or C++ codebase can be compiled into JavaScript using Emscripten, ranging from high performance games that need to render graphics, play sounds, and load and process files, to application frameworks like Qt. Emscripten has already been used to convert a very long list of real-world codebases to JavaScript, including large projects like CPython, Poppler and the Bullet Physics Engine, as well as commercial projects like the Unreal Engine 4 and the Unity engine. Emscripten generates fast code. Its default output format is asm.JS , a highly optimizable subset of JavaScript that can execute close to native speed in many cases [76]. Optimized Emscripten code has also been shown to have a similar effective size to native code, when both are gzipped.

A high level view of the Emscripten toolchain is given below. The main tool is the Emscripten Compiler Frontend (emcc). This is a drop-in replacement for a standard compiler like gcc.

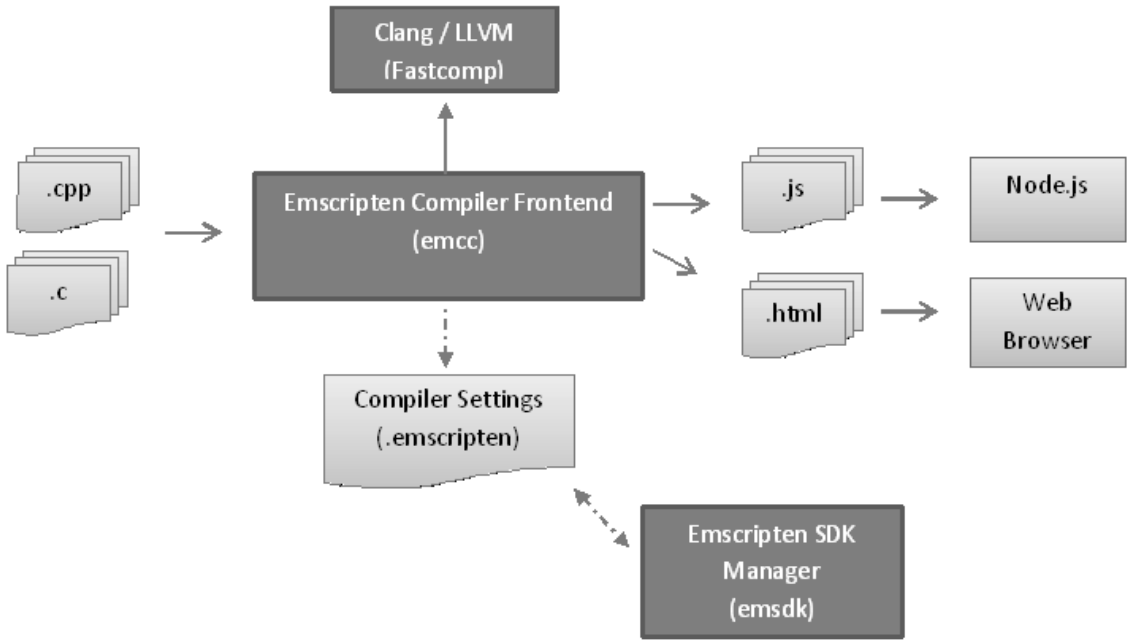


Figure B.1: Emscripten Toolchain

Emcc uses Clang to convert C/C++ files to LLVM bitcode, and Fastcomp (Emscriptenś Compiler Core - an LLVM backend) to compile the bitcode to JavaScript. The output JavaScript can be executed by node.js, or from within HTML in a browser. The Emscripten SDK (emsdk) is used to manage multiple SDKs and tools, and to specify the particular SDK/set of tools currently being used to compile code (the Active Tool/SDK). Emsdk writes the "active" configuration to the Emscripten Compiler Configuration File (.emscripten). This

file is used by emcc to get the correct current toolchain for building. A number of other tools are not shown — for example, Java can optionally be used by emcc to run the closure compiler, which can further decrease code size. The whole toolchain is delivered in the Emscripten SDK, and can be used on Linux, Windows or Mac OS X.

Emscripten support for portable C/C++ code is fairly comprehensive. Support for the C standard library, C++ standard library, C++ exceptions, etc. is very good. OpenGL support in Emscripten support is excellent for OpenGL ES 2.0-type code, and acceptable for other types. There are differences between the native and Emscripten Runtime Environment, which mean some changes usually need to be made to the native code. That said, many applications will only need to change the way they define their main loop, and also modify their file handling to adapt to the limitations of the browser/JavaScript.